globus xio Reference Manual

2.8

Generated by Doxygen 1.2.18

Tue Aug 11 22:41:24 2009

Contents

1 Globus XIO

The Globus eXtensible Input Output library.

- The globus_xio user API.
- User API Assistance.
- Globus XIO Driver
- Driver Programming

2 globus xio Module Index

2.1 globus xio Modules

Here is a list of all modules:

3 globus xio Data Structure Index

3.1 globus xio Data Structures

Here are the data structures with brief descriptions:

globus_xio_http_header_t (HTTP Header) 70

4 globus xio File Index

4.1 globus xio File List

Here is a list of all documented les with brief descriptions:

globus_xio_ le_driver.h (Header le for XIO File Driver) 71
globus_xio_http.h (Globus XIO HTTP Driver Header) 72
globus_xio_mode_e_driver.h (Header le for XIO MODE _E Driver) 74
globus_xio_ordering_driver.h (Header le for XIO ORDERING Driver) 75
globus_xio_tcp_driver.h (Header le for XIO TCP Driver) 75
globus_xio_udp_driver.h (Header le for XIO UDP Driver) 77

5 globus xio Page Index

5.1 globus xio Related Pages

Here is a list of all related documentation pages:

6 globus xio Module Documentation

6.1 The globus_xio user API.

Typedefs

- typedef void(globus_xio_accept_callback_t)(globus_xio_server_t server, globus_xio_handle_t handle, globus_result_t result, void user_arg)
- typedef void(globus_xio_server_callback_t)(globus_xio_server_t server, void user_arg)
- typedef globus_bool_t(globus_xio_timeout_callback_t)(globus_xio_handle_t handle, globus_xio_operation_type_t type, void user_arg)
- typedef void(globus_xio_callback_t)(globus_xio_handle_t handle, globus_result_t result, void user_arg)
- typedef void(globus_xio_data_callback_t)(globus_xio_handle_t handle, globus_result_t result, globus_byte_t buffer, globus_size_t len, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc, void user_arg)
- typedef void(globus_xio_iovec_callback_t)(globus_xio_handle_t handle, globus_result_t result, globus_xio_iovec_t iovec, int count, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc, void user_arg)
- typedef enum globus_i_xio_op_type_e globus_xio_operation_type_t

Enumerations

- enum globus_i_xio_op_type_e
- enum globus_xio_handle_cmd_t { GLOBUS_XIO_GET_LOCAL_CONTACT = 12345, GLOBUS_XIO_GET_LOCAL_NUMERIC_CONTACT, GLOBUS_XIO_GET_REMOTE_CONTACT, GLOBUS_XIO_GET_REMOTE_NUMERIC_CONTACT, GLOBUS_XIO_SEEK, GLOBUS_XIO_SET_STRING_OPTIONS }

Functions

- globus_result_t globus_xio_attr_init (globus_xio_attr_t attr)
- globus_result_t globus_xio_attr_cntl (globus_xio_attr_t attr, globus_xio_driver_t driver, int cmd,...)
- globus_result_t globus_xio_attr_copy (globus_xio_attr_t dst, globus_xio_attr_t src)
- globus_result_t globus_xio_attr_destroy (globus_xio_attr_t attr)
- globus_result_t globus_xio_stack_init (globus_xio_stack_t stack, globus_xio_attr_t stack_attr)
- globus_result_t globus_xio_stack_push_driver (globus_xio_stack_t stack, globus_xio_driver_t driver)
- globus_result_t globus_xio_stack_copy (globus_xio_stack_t dst, globus_xio_stack_t src)
- globus_result_t globus_xio_stack_destroy (globus_xio_stack_t stack)
- globus_result_t globus_xio_server_create (globus_xio_server_t server, globus_xio_attr_t server_attr, globus_xio_stack_t stack)
- globus_result_t globus_xio_server_get_contact_string (globus_xio_server_t server, char contact_string)

globus_result_t globus_xio_server_register_close (globus_xio_server_t server, globus_xio_server_callback_t cb, void user_arg)
globus_result_t globus_xio_server_close (globus_xio_server_t server)
globus_result_t globus_xio_server_cntl (globus_xio_server_t server, globus_xio_driver_t driver, int cmd,...)
globus_result_t globus_xio_server_accept (globus_xio_handle_t out_handle, globus_xio_server_t server)
globus_result_t globus_xio_server_register_accept (globus_xio_server_t server, globus_xio_accept_callback_t cb, void user_arg)
globus_result_t globus_xio_handle_create (globus_xio_handle_t handle, globus_xio_stack_t stack)
globus_result_t globus_xio_data_descriptor_init (globus_xio_data_descriptor_t data_desc, globus_xio_handle_t handle)
globus_result_t globus_xio_data_descriptor_destroy (globus_xio_data_descriptor_t data_desc)
globus_result_t globus_xio_data_descriptor_cntl (globus_xio_data_descriptor_t data_desc, globus_xio_driver_t driver, int cmd,...)
globus_result_t globus_xio_handle_cntl (globus_xio_handle_t handle, globus_xio_driver_t driver, int cmd,...)
globus_result_t globus_xio_register_open (globus_xio_handle_t handle, const char contact_string, globus_xio_attr_t attr, globus_xio_callback_t cb, void user_arg)
globus_result_t globus_xio_open (globus_xio_handle_t handle, const char contact_string, globus_xio_attr_t attr)
globus_result_t globus_xio_register_read (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus_xio_data_callback_t cb, void user_arg)
globus_result_t globus_xio_read (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)
globus_result_t globus_xio_register_readv (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus_xio_iovec_callback_t cb, void user_arg)
globus_result_t globus_xio_readv (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)
globus_result_t globus_xio_register_write (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus_xio_data_callback_t cb, void user_arg)
globus_result_t globus_xio_write (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)
globus_result_t globus_xio_register_writev (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus_xio_iovec_callback_t cb, void user_arg)
globus_result_t globus_xio_writev (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)
globus_result_t globus_xio_register_close (globus_xio_handle_t handle, globus_xio_attr_t attr, globus_xio_callback_t cb, void user_arg)
globus_result_t globus_xio_close (globus_xio_handle_t handle, globus_xio_attr_t attr)
globus_result_t globus_xio_handle_create_from_url (globus_xio_handle_t out_h, const char scheme, globus_xio_attr_t attr, char param_string)
EXTERN_C_END globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_GET_LOCAL_CONTACT, char contact_string_out)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_SEEK, globus_off_t offset)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_SET_STRING_OPTIONS, char con g_string)

6.1.1 Typedef Documentation

6.1.1.1 typedef void(globus_xio_accept_callback_t)(globus_xio_server_t server, globus_xio_handle_t handle, globus_result_t result, void user_arg)

Callback signature for accept.

When a registered accept operation completes the users function of this signature is called.

Parameters:

- server The server object on which the accept was registered.
- handle The newly created handle that was created by the accept operation.
- result A result code indicating the success of the accept operation. GLOBUS_SUCCESS indicates a successful accept.
- user_arg A user argument that is threaded from the registration to the callback.

6.1.1.2 typedef void(globus_xio_server_callback_t)(globus_xio_server_t server, void user_arg)

Server callback signature.

This is the generic server callback signature. It is currently only used for the register close operation.

6.1.1.3 typedef globus_bool_t(globus_xio_timeout_callback_t)(globus_xio_handle_t handle, globus_xio_operation_type_t type, void user_arg)

The timeout callback function signature.

Parameters:

- handle The handle the handle on which the timeout operation was requested.
- type The type of opperation that timed out: GLOBUS_XIO_OPERATION_OPEN GLOBUS_XIO_OPERATION_CLOSE GLOBUS_XIO_OPERATION_READ GLOBUS_XIO_OPERATION_WRITE
- arg A user arg threaded throw to the callback.

6.1.1.4 typedef void(globus_xio_callback_t)(globus_xio_handle_t handle, globus_result_t result, void user_arg)

globus_xio_callback_t

This callback is used for the open and close asynchronous operations.

6.1.1.5 typedef void(globus_xio_data_callback_t)(globus_xio_handle_t handle, globus_result_t result, globus_byte_t buffer, globus_size_t len, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc, void user_arg)

globus_xio_data_callback_t

This callback is used for asychronous operations that send or receive data.

on eof, result_t will be of type GLOBUS_XIO_ERROR_EOF

6.1.1.6 typedef void(globus_xio_iovec_callback_t)(globus_xio_handle_t handle, globus_result_t result, globus_xio_iovec_t iovec, int count, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc, void user_arg)

globus_xio_iovec_callback_t

This callback is used for asychronous operations that send or receive data with an ivec structure.

on eof, result_t will be of type GLOBUS_XIO_ERROR_EOF

6.1.1.7 typedef enum globus_i_xio_op_type_e globus_xio_operation_type_t

Operation types.

An enumeration of operation types. Used in the timeout callback to indicate what operation typed timedout.

6.1.2 Enumeration Type Documentation

6.1.2.1 enum globus_i_xio_op_type_e

Operation types.

An enumeration of operation types. Used in the timeout callback to indicate what operation typed timedout.

6.1.2.2 enum globus_xio_handle_cmd_t

Common driver handle cntls.

Enumeration values:

GLOBUS_XIO_GET_LOCAL_CONTACT	See usage for globus_xio_handle_cntl.
GLOBUS_XIO_GET_LOCAL_NUMERIC_CONTACT	See usage for globus_xio_handle_cntl.
GLOBUS_XIO_GET_REMOTE_CONTACT	See usage for globus_xio_handle_cntl.
GLOBUS_XIO_GET_REMOTE_NUMERIC_CONTACT	See usage for globus_xio_handle_cntl.
GLOBUS_XIO_SEEK	See usage for globus_xio_handle_cntl.
GLOBUS_XIO_SET_STRING_OPTIONS	See usage for globus_xio_handle_cntl.

6.1.3 Function Documentation

6.1.3.1 globus_result_t globus_xio_attr_init (globus_xio_attr_t attr)

Intialize a globus xio attribute.

Parameters:

attr	upon return from this function this out parameter will be initialized. Once the user is finished with the attribute they should make sure they destroy it in order to free resources associated with it.

6.1.3.2 globus_result_t globus_xio_attr_cntl (globus_xio_attr_t attr, globus_xio_driver_t driver, int cmd, ...)

Manipulate the values associated in the attr.

This function provides a means to access the attr structure. What exactly this function does is determined by the value in the parameter cmd and the value of the patameter driver. When the driver parameter is NULL it indicates that this function applies to general globus xio values. If it is not NULL it indicates that the function will effect driver specific values. Each driver is resonsible for defining its own enumeration of values for cmd and the var args associated with that command.

Parameters:

- attr the attribute structure to be manipulated.
- driver This parameter indicates which driver the user would like to perform the requested operation. If this parameter is NULL this request will be scoped to general attribure functions.
- cmd an enum that determines what specific operation the user is requesting. Each driver will determine the value for this enumeration.

6.1.3.3 globus_result_t globus_xio_attr_copy (globus_xio_attr_t * dst, globus_xio_attr_t src)

Copy an attribute structure.

6.1.3.4 globus_result_t globus_xio_attr_destroy (globus_xio_attr_t attr)

Clean up resources associated with an attribute.

Parameters:

- attr Upon completion of this function all resources associated with this structure will returned to the system and the attr will no longer be valid.

6.1.3.5 globus_result_t globus_xio_stack_init (globus_xio_stack_t * stack, globus_xio_attr_t stack_attr)

Initialize a stack object.

6.1.3.6 globus_result_t globus_xio_stack_push_driver (globus_xio_stack_t stack, globus_xio_driver_t driver)

Push a driver onto a stack.

No attrs are associated with a driver. The stack represents the ordered lists of transform drivers and 1 transport driver. The transport driver must be pushed on first.

6.1.3.7 globus_result_t globus_xio_stack_copy (globus_xio_stack_t * dst, globus_xio_stack_t src)

Copy a stack object.

6.1.3.8 globus_result_t globus_xio_stack_destroy (globus_xio_stack_t stack)

Destroy a stack object.

6.1.3.9 globus_result_t globus_xio_server_create (globus_xio_server_t * server, globus_xio_attr_t server_attr, globus_xio_stack_t stack)

Create a server object.

This function allows the user to create a server object which can then be used to accept connections.

Parameters:

- server An out parameter. Once the function successfully returns this will point to a valid server object.
- server_attr an attributre structure used to alter the default server intialization. This will mostly be used in a driver specific manner. can be NULL.
- stack

6.1.3.10 globus_result_t globus_xio_server_get_contact_string (globus_xio_server_t server, char ** contact_string)

get contact string

This function allows the user to get the contact string for a server. this string could be used as the contact string for the client side.

Parameters:

- **server** An initialized server handle created with globus_xio_server_create()
- **contact_string** an out varibale. Will point to a newly allocated string on success. must be freed by the caller.

6.1.3.11 globus_result_t globus_xio_server_register_close (globus_xio_server_t server, globus_xio_server_callback_t cb, void * user_arg)

post a close on a server object

This function registers a close operation on a server. When the user function pointed to by parameter cb is called the server object is closed.

6.1.3.12 globus_result_t globus_xio_server_close (globus_xio_server_t server)

A blocking server close.

6.1.3.13 globus_result_t globus_xio_server_cntl (globus_xio_server_t server, globus_xio_driver_t driver, int cmd, ...)

Touch driver specific information in a server object.

This function allows the user to comunicate directly with a driver in association with a server object. The driver defines what operations can be preformed.

6.1.3.14 globus_result_t globus_xio_server_accept (globus_xio_handle_t * out_handle, globus_xio_server_t server)

Accept a connection.

This function will accept a connetion on the given server object and the parameter out_handle will be valid if the function returns successfully.

6.1.3.15 globus_result_t globus_xio_server_register_accept (globus_xio_server_t server, globus_xio_accept_callback_t cb, void * user_arg)

Asynchronous accept.

This function posts an nonblocking accept. Once the operation has completed the user function pointed to by the parameter cb is called.

6.1.3.16 globus_result_t globus_xio_handle_create (globus_xio_handle_t * handle, globus_xio_stack_t stack)

Initialize a handle for client opens.

This funtion will initialize a handle for active opens (client side connections).

6.1.3.17 globus_result_t globus_xio_data_descriptor_init (globus_xio_data_descriptor_t * data_desc, globus_xio_handle_t handle)

Initialize a data descriptor.

Parameters:

- data_desc An out parameter. The data descriptor to be intialized.
- handle The handle this data descriptor will be used with. This parametter is require in order to optimize the code handling the data descriptors use.

6.1.3.18 globus_result_t globus_xio_data_descriptor_destroy (globus_xio_data_descriptor_t data_desc)

clean up a data descriptor.

6.1.3.19 globus_result_t globus_xio_data_descriptor_cntl (globus_xio_data_descriptor_t data_desc, globus_xio_driver_t driver, int cmd, ...)

Touch driver specific data in data descriptors.

This function allows the user to comunicate directly with a driver in association with a data descriptors. The driver defines what operations can be preformed.

6.1.3.20 globus_result_t globus_xio_handle_cntl (globus_xio_handle_t handle, globus_xio_driver_t driver, int cmd, ...)

Touch driver specific information in a handle object.

This function allows the user to comunicate directly with a driver in association with a handle object. The driver defines what operations can be preformed.

pass the driver to control a specific driver pass NULL for driver for XIO specific cntls pass GLOBUS_XIO_QUERY for driver to try each driver in order until success

6.1.3.21 globus_result_t globus_xio_register_open (globus_xio_handle_t handle, const char * contact_string, globus_xio_attr_t attr, globus_xio_callback_t cb, void * user_arg)

Open a handle.

Creates an open handle based on the state contained in the given stack.

No operation can be preformed on a handle until it is initialized and then opened. If an already open handle used the information contaned in that handle will be destoyed.

Parameters:

- handle The handle created with globus_xio_handle_create() or globus_xio_server_register_accept() that is to be opened.
- attr how to open attribute. can be NULL
- cb The function to be called when the open operation completes.
- user_arg A user pointer that will be threaded through to the callback.
- contact_string An url describing the resource. NULL is allowed. Drivers interpret the various parts of this url as descibed in their documentation. An alternative form is also supported: if contact_string does not specify a scheme (e.g. http://) and it contains a ':', it will be parsed as a host:port pair. if it does not contain a ':', it will be parsed as the path

the following are examples of valid formats:

```
<path  to  file>
host-name  ":"  <service  or  port>
"file:"  <path  to  file>
<scheme>  "://"  [  "/"  [  <path  to  resource>  ]    ]
<scheme>  "://"  location  [  "/"  [  <path  to  resource>  ]  ]
   location:
         [  auth-part  ]  host-part
   auth-part:
         <user>  [  ":"  <password>  ]  "@"
   host-part:
         [  "<"  <subject>  ">"  ]  host-name  [  ":"  <port  or  service>  ]
   host-name:
         <hostname>  |  <dotted  quad>  |  "["  <ipv6  address>  "]"
```

Except for use as the above delimeters, the following special characters MUST be encoded with the %HH format where H == hex char.

```
"/"  and  "@"  in  location  except  subject
"<"  and  ">"  in  location
":"  everywhere  except  ipv6  address  and  subject
"%"  everywhere  (can  be  encoded  with  %HH  or  %%)
```

6.1.3.22 globus_result_t globus_xio_open (globus_xio_handle_t handle, const char contact_string, globus_xio_attr_t attr)

blocking open

6.1.3.23 globus_result_t globus_xio_register_read (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus_xio_data_callback_t cb, void user_arg)

Read data from a handle.

6.1.3.24 globus_result_t globus_xio_read (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)

Read data from a handle.

6.1.3.25 globus_result_t globus_xio_register_readv (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus_xio_iovec_callback_t cb, void user_arg)

Read data from a handle into a globus_xio_iovec_t (struct iovec).

6.1.3.26 globus_result_t globus_xio_readv (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)

Read data from a handle into a globus_xio_iovec_t (struct iovec).

6.1.3.27 globus_result_t globus_xio_register_write (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus_xio_data_callback_t cb, void user_arg)

Write data to a handle.

6.1.3.28 globus_result_t globus_xio_write (globus_xio_handle_t handle, globus_byte_t buffer, globus_size_t buffer_length, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)

Write data to a handle.

6.1.3.29 globus_result_t globus_xio_register_writev (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_xio_data_descriptor_t data_desc, globus_xio_iovec_callback_t cb, void user_arg)

Write data to a handle from a globus_xio_iovec_t (struct iovec).

6.1.3.30 globus_result_t globus_xio_writev (globus_xio_handle_t handle, globus_xio_iovec_t iovec, int iovec_count, globus_size_t waitforbytes, globus_size_t nbytes, globus_xio_data_descriptor_t data_desc)

Write data to a handle from a globus_xio_iovec_t (struct iovec).

6.1.3.31 globus_result_t globus_xio_register_close (globus_xio_handle_t handle, globus_xio_attr_t attr, globus_xio_callback_t cb, void user_arg)

Close a handle.

This functions servers as a destoy for the handle. As soon as the operations completes (the callback is called). The handle is destroyed.

Parameters:
- handle the handle to be closed.
- attr how to close attribute
- cb The function to be called when the close operation completes.
- user_arg A user pointer that will be threaded through to the callback.

6.1.3.32 globus_result_t globus_xio_close (globus_xio_handle_t handle, globus_xio_attr_t attr)

Blocking close.

6.1.3.33 globus_result_t globus_xio_handle_create_from_url (globus_xio_handle_t out_h, const char scheme, globus_xio_attr_t attr, char param_string)

Initializes a handle based on the scheme given.

Parameters:
- out_h An uninitialized handle that will be initialized in the function to correspond to the scheme given. This handle should be used for any I/O operations.
- scheme A string containing the protocol which the handle should be initialized to. The string can either be a protocol by itself, for example, "http", or a complete scheme such as "http://www.example.com".
- attr Attribute to be used for setting parameter string. It is initialized by the function. Can be NULL if attributes are not being used.

param_string A string containing attributes to be set for the drivers associated with the scheme. This should be in the form "protocol1:option1=value1;option2=value2,protocol2:option1=value1; option2=value2" Can be NULL if attributes are not being used.

6.1.3.34 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_GET_LOCAL_CONTACT, char ∗∗ contact_string_out)

Get local connection info.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

contact_string_out A pointer to a contact string for the local end of a connected handle. Where possible, it will be in symbolic form (FQDN).

The user must free the returned string.

See also:

globus_xio_server_get_contact_string()

6.1.3.35 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_SEEK, globus_off_t offset)

Reposition read/write offset.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

offset Specify the desired offset.

6.1.3.36 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_SET_STRING_OPTIONS, char ∗ con g_string)

Set the driver speci c con guration string.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

The format of the string is de ned by the driver. It is typically a set of key=value pairs

Parameters:

con g_string The driver speci c paramter string.

6.2 User API Assistance.

Help understanding the globus_xio api.

Stack Constuction.

The driver stack that is used for a given xio handle is constructed using a globus_xio_stack_t. Each driver is loaded by name and pushed onto a stack.

```
stack setup example:

// First load the drivers
globus_xio_driver_load("tcp", &tcp_driver);
globus_xio_driver_load("gsi", &gsi_driver);

//build the stack
globus_xio_stack_init(&stack);
globus_xio_stack_push_driver(stack, tcp_driver, NULL);
globus_xio_stack_push_driver(stack, gsi_driver, NULL);
```

Servers

A server data structure provides functionality for passive opens. A server is initialized and bound to a protocol stack and set of attributes with the function globus_xio_server_create(). Once a server is created many "connections" can be accepted. Each connection will result in an intialized handle which can later be opened.

```
globus_xio_server_t              server;
globus_xio_attr_t                attr;

globus_xio_attr_init(&attr);
globus_xio_server_create(&server_handle, attr, stack);
globus_xio_server_accept(&handle, server);
```

Handle Construction

There are two ways to create a handle. The first is for use as a client (one that is doing an active open). The function: globus_xio_handle_create() is used to create such a handle and bind that handle to a protocol stack.

```
globus_xio_handle_create(&handle, stack);
```

The second means of creating a handle is for use as a server (one that is doing a passive open). This is created by accepting a connection on a server_handle with the function globus_xio_server_accept() or globus_xio_server_register_accept().

Mutable attrs can be altered via a call to globus_xio_handle_cntl() described later.

```
globus_xio_server_accept(&xio_handle, server_handle);
```

once a handle is intialized the user can call globus_xio_open() to begin the open process.

Timeouts

A user can set a timeout value for any io operation. Each IO operation (open close read write) can have its own timeout value. If no timeout is set the operation will be allowed to infinitly block.
When time expires the outstanding operation is canceled. If the timeout callback for the given operation is not NULL it is called first to notify the user that the operation timed out and give the user a chance to ignore that timeout. If canceled, the user will get the callback they registered for the operation as well, but it will come with an error indicating that it has been canceled.
It is possiblie that part of an io operation will complete before the timeout expires. In this case the opperation can still be canceled. The user will receive there IO callback with and error set and the length value appropriately set to indicate how much of the operation completed.

Data Desciptor

The data descriptor ADT gives the user a means of attaching/extracting meta data to a read or write operation.
Things like offset, out of band message, and other driver specific meta data are contained in the data descriptor.
Data descriptors are passed to globus_xio in globus_xio_read() and globus_xio_write(). Within the globus_xio framework it is acceptable to pass NULL instead of a valid data_descriptor,

```
ex:
globus_xio_data_descriptor_init(&desc);
globus_xio_data_descriptor_cntl(desc,
        tcp_driver,
        GLOBUS_XIO_TCP_SET_SEND_FLAGS,
        GLOBUS_XIO_TCP_SEND_OOB);
```

User Attributes

Globus XIO uses a single attribute object for all of its functions. Attributes give an the user an extenable mechanism to alter default values which control parameters in an operation.

In most of the globus xio user api functions a user passes an attribute as a parameter. In many cases the user may ignore the attribute parameter and just pass in NULL. However at times the user will wish to tweak the operation. The attribute structure is used for this tweaking.

There are only three attribute functions. The globus xio user API.Attributes and Cntls and The globus xio user API.. The init and destroy functions are very simple and require little explaination. Before an attribute can be used it must be intialized, and to clean up all memory associated with it the user must call destroy on it.

The function Attributes and Cntls manipulates values in the attribute. For more info on it see Attributes and Cntls.

6.3 Globus XIO Driver

Globus XIO introduces a notion of a driver stack to its API. With in globus xio every IO operation must occur on a globus xio handle. Associated with each handle is a stack of drivers. A driver is a module piece of code that implements the globus xio driver interface. The purpose of a driver is manipulate data passed in by the user in someway. Each driver in a stack will serve its own unique purpose.

IO operations pass from driver to driver, starting at the top of the stack and ending at the bottom. When the bottom layer driver nishes with the operation it signals globus xio that it has completed. Completion noti cation then follows the driver stack up to the top.

Driver Types:

Transport driver:

A transport driver is one that is responsible for actually putting bytes onto the wire. For example: A TCP driver or a UDP driver would be an example of transport drivers.

Per driver stack there must be exactly one transport driver and must be at the bottom of the stack. A transform driver is de ned by its lack of passing an operation to the next driver in the stack. This type of driver does not rely on globus xio for further completion of an operation, rather it is self suf cent in this task.

Transform driver:

A tranform driver is any intermediate driver in the stack. These drivers are indenti ed by there reliance on the driver stack to complete the operation. These drivers must pass the operation down the stack because they cannot complete it themselves. An example of a transform driver would be a gsi driver. This driver would wrap and unwrap messages, but would not be able to complete the transport itself, so it would rely on the remaining drivers in the stack.

Driver API

The globus xio driver api is a set of functions and interfaces to allow a developer to create a backend driver for globus xio. To create a driver the user must implement all of the interface functions in the driver speci cation. There are also a set of functions provide to assist the driver author in implemention.

Quick Start:

Four basic driver needs the user will have to pay attention to a few new structures and concepts.

globus_xio_operation_t

This structure represents a request for an operation. If the driver can service the operation it does so and the calls the appropriate nish operation() function. If the driver cannot completely service the operation it can pass() it along to the next driver in the stack. As soon as the operation structure is either nished or passed it is no longer valid for use in any other function.

globus_xio_driver_handle_t
A driver_handle represents a open handle to the driver stack for xio. The driver obtains a driver handle by calling globus_xio_driver_open(). When the open operation completes (it callback is called) the driver then has a driver handle. The driver_handle allows the user to do some complex things that will be described later.
globus_xio_stack_t
This structure provides the driver with information about the driver stack It is mainly used for creating a driver handle as a parameter to lobus_xio_driver_open()..

Typical Sequence:
Here is a typcial sequence of events for a globus_xio transform driver:
Open
globus_xio_driver_open_t is called. The user calls globus_xio_driver_open() passing it the operation and the stack and a callback. When the open callback is called the driver is given a new operation as a parameter. The driver will then call globus_xio_driver_ nished_open() passing it the now initialized driver_handle and the newly received operation. The call to globus_xio_driver_ nished_open() does two things: 1) it tells globus_xio that this driver has nished its open operation, and 2) it gives xio the driver_handle (which contains information on the drivers below it).
Read/Write
The read or write interface funcion is called. It receives a operation as a parameter. The driver then calls the approriate pass operation and waits for the callback. When the callback is received the driver calls _ nished operation passing in the operation structure it received in the callback
Close
The close interface function is called and is passed an operation and a driver_handle. The driver will call globus_xio_driver_close() passing it the operation. When the close callback is received the driver calls globus_xio_driver_ nished_close() passing it the operation received in the close callback and the driver_handle received in the interface function. At this point the driver_handle is no longer valid..

Advanced Driver Programming
The typical driver implementatin is describe above. However globus_xio allows driver authors to do more advanced things. Some of these things will be explored here.
Read Ahead
Once a driver_handle is open a driver can spawn operation structures from it. This gives the driver the ability to request io from the driver stack before it receives a call to its own interface io interface function. So if a driver wishes to read ahead it does the following:

it creats an operation by calling globus_xio_driver_create_operation() and passing it the driver_handle it is intereesting in using.

call globus_xio_driver_read() using this operations. When the read callback is received the driver may call nished_operation() on the op it receives (this ultimitely results in very little, since this operation was started by this driver, but it is good practice and will free up resources that would otherwise leak).

Now when the user nally does receive a read interface call from globus_xio it can imediately nish it using the operation it just received as a parameter and updating the iovec structure to represent the read that already occured.

Preopening handles.

Once the driver has received a globus_xio_driver_stack_t it can open a driver_handle. The globus_xio_driver_stack_t comes in the call to the interface function globus_xio_server/client_init_t(). The driver uses this structure in a call to globus_xio_driver_open(). When this functionality completes the driver has an initialized driver handle and can use it to create operations as described above. The driver can now hang onto this driver handle until it receives an open interface function call. At which time it can call globus_xio_driver_ nished_open() passing in the driver_handle and thereby glueing the pre opened driver_handle with the requested globus_xio operation.

6.4 Driver Programming

The set of interface functions that the driver author must implement to create a driver and the functions to assist in the creation.

Typedefs

typedef void(globus xio_driver_callback t)(globus xio_operation t op, globus result t result, void user arg)
typedef void(globus xio_driver_data callback t)(globus xio_operation t op, globus result t result, globus size t nbytes, void user arg)
typedef globus result t(globus xio_driver_attr_init_t)(void out_driver_attr)
typedef globus result t(globus xio_driver_attr_copy_t)(void dst, void src)
typedef globus result t(globus xio_driver_attr_destroy t)(void driver_attr)
typedef globus result t(globus xio_driver_attr_cntl_t)(void attr, int cmd, va list ap)
typedef globus result t(globus xio_driver_server init_t)(void driver_attr, const globus xio_contact-t contact info, globus xio_operation t op)
typedef globus result t(globus xio_driver_server destroy t)(void driver_server)
typedef globus result t(globus xio_driver_server accept t)(void driver_server, globus xio_operation t op)
typedef globus result t(globus xio_driver_server cntl_t)(void driver_server, int cmd, va list ap)
typedef globus result t(globus xio_driver_link_destroy t)(void driver_link)
typedef globus result t(globus xio_driver_transform open t)(const globus xio_contact t contact info, void driver_link, void driver_attr, globus xio_operation t op)
typedef globus result t(globus xio_driver_transport open t)(const globus xio_contact t contact info, void driver_link, void driver_attr, globus xio_operation t op)
typedef globus result t(globus xio_driver_handle cntl_t)(void handle, int cmd, va list ap)
typedef globus result t(globus xio_driver_close t)(void driver_handle, void driver_attr, globus xio_-operation t op)
typedef globus result t(globus xio_driver_read t)(void driver_speci c_handle, const globus xio_iovec_-t iovec, int iovec count, globus xio_operation t op)
typedef globus result t(globus xio_driver_write_t)(void driver_speci c_handle, const globus xio_iovec_-t iovec, int iovec count, globus xio_operation t op)

Functions

globus result t globus xio_driver_handle cntl (globus xio_driver_handle t driver_handle, globus xio_driver_-t driver, int cmd,...)
void globus xio_driver_ nished_accept(globus xio_operation t op, void driver_link, globus result t result)
globus result t globus xio_driver_pass open(globus xio_operation t op, const globus xio_contact t contact-info, globus xio_driver_callback t cb, void user arg)
void globus xio_driver_ nished_open(void driver_handle, globus xio_operation t op, globus result t result)
globus result t globus xio_driver_operation create (globus xio_operation t operation, globus xio_driver_-handle t driver_handle)
globus bool_t globus xio_driver_operation is_blocking(globus xio_operation t op)
globus result t globus xio_driver_pass close(globus xio_operation t op, globus xio_driver_callback t cb, void user arg)
void globus xio_driver_ nished_close(globus xio_operation t op, globus result t result)
globus result t globus xio_driver_pass read(globus xio_operation t op, globus xio_iovec t iovec, int iovec-count, globus size t wait_for, globus xio_driver_data callback t cb, void user arg)
void globus xio_driver_ nished_read(globus xio_operation t op, globus result t result, globus size t nread)
void globus xio_driver_set eof_received(globus xio_operation t op)

globus_bool_t globus_xio_driver_eof_received(globus_xio_operation_t op)
globus_result_t globus_xio_driver_pass_write (globus_xio_operation_t op, globus_xio_iovec_t iovec, int iovec_count, globus_size_t wait_for, globus_xio_driver_data_callback_t cb, void user_arg)
void globus_xio_driver_ nished_write (globus_xio_operation_t op, globus_result_t result, globus_size_t nwritten)
globus_result_t globus_xio_driver_merge_operation(globus_xio_operation_t top_op, globus_xio_operation_t bottom_op)

6.4.1 Detailed Description

The set of interface functions that the driver author must implement to create a driver and the functions to assist in the creation.

Driver attribute functions

If the driver wishes to provide driver speci c attributes to the user it must implement the following functions:

globus_xio_driver_attr_init_t globus_xio_driver_attr_copy_t globus_xio_driver_attr_cntl_t globus_xio_driver_attr_destroy_t

6.4.2 Typedef Documentation

6.4.2.1 typedef void(globus_xio_driver _callback_t)(globus_xio_operation_t op, globus_result_t result, void user_arg)

callback interface

This is the function signature of callbacks for the globus_xio_driver_open/close().

Parameters:

op The operation structure associated with the open or the close requested operation. The driver should call the appropriate nished operation to clean up this structure.

result The result of the requested data operation

user_arg The user pointer that is threaded through to the callback.

6.4.2.2 typedef void(globus_xio_driver _data_callback_t)(globus_xio_operation_t op, globus_result_t result, globus_size_t nbytes, void user_arg)

Data Callback interface.

This is the function signature of read and write operation callbacks.

Parameters:

op The operation structure associated with the read or write operation request. The driver should call the appropriate nished operation when it receives this operation.

result The result of the requested data operation

nbytes the number of bytes read or written

user_arg The user pointer that is threaded through to the callback.

6.4.2.3 typedef globus_result_t(globus_xio_driver _attr _init _t)(void out_driver _attr)

Create a driver speci c attribute.

The driver should implement this function to create a driver speci c attribute and return it via the out_attr parameter.

6.4.2.4 typedef globus_result_t(globus_xio_driver_attr_copy_t)(void dst, void src)

Copy a driver attr.

When this function is called the driver will create a copy of the attr in parameter src and place it in the parameter dst.

6.4.2.5 typedef globus_result_t(globus_xio_driver_attr_destroy_t)(void driver_attr)

Destroy the driver attr.

Clean up all resources associate with the attr.

6.4.2.6 typedef globus_result_t(globus_xio_driver_attr_cntl_t)(void attr, int cmd, va_list ap)

get or set information in an attr.

The cmd parameter determines what functionality the user is requesting. The driver is resonsible for providing documentation to the user on all the possible values that cmd can be.

Parameters:

driver_attr The driver speci c attr, created by globus_xio_driver_attr_init_t.

cmd An integer representing what functionality the user is requesting.

ap variable arguments. These are determined by the driver and the value of cmd.

6.4.2.7 typedef globus_result_t(globus_xio_driver_server_init_t)(void driver_attr, const globus_xio_contact_t contact_info, globus_xio_operation_t op)

Initialize a server object.

The driver developer should implement this function if their driver handles server operations (pasive opens). In the tcp driver this function should create a listener.

Parameters:

op An op which should be passed to globus_xio_driver_server_created. Note, that unlike most operations, the server is created from the bottom of the stack to the top.

contact_info This the contact info for the stack below this driver. (entries will always be NULL for the transport driver)

driver_attr A server attr if the user speci ed any driver speci c attributes. This may be NULL.

Returns:

Returning GLOBUS_SUCCESS for this means that ` globus_xio_driver_pass_server_init returned success and the driver's server was successfully initialized.

6.4.2.8 typedef globus_result_t(globus_xio_driver_server_destroy_t)(void driver_server)

destroy a server.

When this function is called the driver should free up all resources associated with a server.

Parameters:

server The server that the driver should clean up.

driver_server The reference to the iunternal server that is being declaired invaild with this function call.

6.4.2.9 typedef globus_result_t(globus_xio_driver_server_accept_t)(void driver_server, globus_xio_operation_t op)

Accept a server connection.

The driver developer should implement this function if their driver handles server operations. Once the accept operation completes, the connection is established. The user still has an opertunity to open the link or destroy it. They can query the link for additional information on which to base the decision to open.

Parameters:

- driver_server The server object from which the link connection will be accepted.
- op The requested operation. When the driver is nished acepting the server connection it uses this structure to signal globus_xio that it has completed the operation.

6.4.2.10 typedef globus_result_t(globus_xio_driver_server_cntl_t)(void driver_server, int cmd, va_list ap)

Query a server for information.

This function allows a user to request information from a driver speci c server handle.

Parameters:

- driver_server the server handle.
- cmd An integer telling the driver what operation to preform on this server handle.
- ap variable args.

6.4.2.11 typedef globus_result_t(globus_xio_driver_link_destroy_t)(void driver_link)

destroy a link

The driver should clean up all resources associated with the link when this function is called.

Parameters:

- driver_link The link to be destroyed.

6.4.2.12 typedef globus_result_t(globus_xio_driver_transform_open_t)(const globus_xio_contact_t contact_info, void driver_link, void driver_attr, globus_xio_operation_t op)

Open a handle.

This is called when a user requests to open a handle.

Parameters:

- driver_link Comes from server accept. Used to link an accepted connection to an xio handle. xio will destroy this object upon the return of this interface call.
- driver_attr A attribute describing how to open. This points to a piece of memory created by the globus_xio_driver_driver_attr_init_t interface funstion.
- contact_info Contains information about the requested resource. Its members may all be null (especially when link is not null). XIO will destroy this contact info upon return from the interface function
- op The requested operation. When the driver is nished opening the handle it uses this structure to signal globus_xio that it has completed the operation requested. It does this by calling globus_xio_driver_ nished_open()

6.4.2.13 typedef globus_result_t(globus_xio_driver_transport_open_t)(const globus_xio_contact_t contact_info, void driver_link, void driver_attr, globus_xio_operation_t op)

transport open

6.4.2.14 typedef globus_result_t(globus_xio_driver_handle_cntl_t)(void handle, int cmd, va_list ap)

this call must return an GLOBUS_XIO_ERROR_COMMAND error for unsupported command numbers.

(use GlobusXIOErrorInvalidCommand(cmd))

Drivers that have reason to support the commands listed at The globus_xio user API. should accept the xio generic cmd numbers and their driver speci c command number. Do NOT implement those handle cntls unless you really are the de nitive source.

6.4.2.15 typedef globus_result_t(globus_xio_driver_close_t)(void driver_handle, void driver_attr, globus_xio_operation_t op)

Close an open handle.

This is called when a user requests to close a handle. The driver implemntor should clean up all resources connected to there driver handle when this function is called.

Parameters:
- driver_speci c_handle The driver handle to be closed.
- driver_attr A driver speci c attr which may be used to alter how a close is performed (e,g, caching drivers)
- op The requested operation. When the driver is nished closing the handle it uses this structure to signal globus_xio that it has completed the operation requested. It does this by calling globus_xio_driver_ nished_close()

6.4.2.16 typedef globus_result_t(globus_xio_driver_read_t)(void driver_speci c_handle, const globus_xio_iovec_t iovec, int iovec_count, globus_xio_operation_t op)

Read data from an open handle.

This function is called when the user requests to read data from a handle. The driver author shall implement all code needed to for there driver to complete a read operations.

Parameters:
- driver_handle The driver handle from which data should be read.
- iovec An io vector pointing to the buffers to be read into.
- iovec_count The number if entries in the io vector.
- op The requested operation. When the driver is nished full lling the requested read operation it must use this structure to signal globus_xio that the operation is completed. This is done by calling globus_xio_driver_nished_operation()..

6.4.2.17 typedef globus_result_t(globus_xio_driver_write_t)(void driver_speci c_handle, const globus_xio_iovec_t iovec, int iovec_count, globus_xio_operation_t op)

Write data from an open handle.

This function is called when the user requests to write data to a handle. The driver author shall implement all code needed to for there driver to complete write operations.

Parameters:

- **driver_handle** The driver handle to which data should be writen.
- **iovec** An io vector pointing to the buffers to be written.
- **iovec_count** The number if entries in the io vector.
- **op** The requested operation. When the driver is nished full lling the requested read operation it must use this structure to signal globus_xio that the operation is completed. This is done by calling globus_xio_driver_- nished_operation()..

6.4.3 Function Documentation

6.4.3.1 globus_result_t globus_xio_driver_handle_cntl (globus_xio_driver_handle_t *handle*, globus_xio_driver_t *driver*, int *cmd*, ...)

Touch driver speci c information in a handle object.

pass the driver to control a speci c driver pass NULL for driver for XIO speci c cntls pass GLOBUS_XIO_QUERY for driver to try each driver (below current) in order

6.4.3.2 void globus_xio_driver_ nished_accept (globus_xio_operation_t *op*, void * *driver_link*, globus_result_t *result*)

Driver API nished accept.

This function should be called to signal globus_xio that it has completed the accept operation requested of it. It will free up resources associated with the accept_op and potientially cause xio to pop the signal up the driver stack.

Parameters:

- **op** The requested accept operation that has completed.
- **driver_link** This is the initialized driver link that is that will be passed to the open interface when this handle is opened.
- **result** Return status of the completed operation

6.4.3.3 globus_result_t globus_xio_driver_pass_open (globus_xio_operation_t *op*, const globus_xio_contact_t * *contact_info*, globus_xio_driver_callback_t *cb*, void * *user_arg*)

Driver API Open.

This function will pass an open request down the driver stack. Upon completion of the open operation globus_xio will call the cb function, at which point the handle structure will be intialized and available for use.

As soon as the function returns the handle is valid for creating other operations.

Parameters:

- **op** The operation from which the handle will be established. This parameter is used to determine what drivers are in the stack and other such information.
- **contact_info** The contact info describing the resource the driver below should open. This will normally be the same contact info that was passed in on the open interface.
- **cb** The function to be called wehn the open operation is complete.
- **user_arg** a user pointer that will be threaded through to the callback.

6.4.3.4 void globus_xio_driver_finished_open (void * driver_handle, globus_xio_operation_t op, globus_result_t result)

Driver API finished open.

This function should be called to signal globus_xio that it has completed the open operation requested of it. It will free up resources associated with the op and potientially cause xio to pop the signal up the driver stack.

Parameters:

- **driver_handle** The driver specific handle pointer that will be passed to future interface funstion calls.
- **op** The requested open operation that has completed.
- **result** Return status of the completed operation

6.4.3.5 globus_result_t globus_xio_driver_operation_create (globus_xio_operation_t * operation, globus_xio_driver_handle_t handle)

Driver API Create Operation.

This function will create an operation from an initialized handle This operation can then be used for io operations related to the handle that created them.

Parameters:

- **operation** The operation to be created. When this function returns this structure will be populated and available for use for the driver.
- **handle** The initialized handle representing the user handle from which the operation will be created.

6.4.3.6 globus_bool_t globus_xio_driver_operation_is_blocking (globus_xio_operation_t operation)

Is Operation blocking.

If the operation is blocking the driver developer may be able to make certian optimizations. The function returns true if the given operation was created via a user call to a blocking funciton.

6.4.3.7 globus_result_t globus_xio_driver_pass_close (globus_xio_operation_t op, globus_xio_driver_callback_t cb, void * user_arg)

Driver API Close.

This function will pass a close request down the driver stack. Upon completion of the close operation globus_xio will call the funciton pointed to by the cb arguement.

Parameters:

- **op** The operation to pass along the driver stack for closing.
- **cb** A pointer to the function to be called once all drivers lower in the stack have closed.
- **user_arg** A user pointer that will be threaded through to the callback.

6.4.3.8 void globus_xio_driver_finished_close (globus_xio_operation_t op, globus_result_t result)

Driver API finished_close.

The driver calls this function after completing a close operation on a driver_handle. Once this function returns the driver_handle is no longer valid.

Parameters:

op The close operation that has completed.

result Return status of the completed operation

6.4.3.9 globus_result_t globus_xio_driver_pass_read (globus_xio_operation_t op, globus_xio_iovec_t * iovec, int iovec_count, globus_size_t wait_for, globus_xio_driver_data_callback_t cb, void * user_arg)

Driver read.

This function passes a read operation down the driver stack. After this function is called the op structure is no longer valid. However when the driver stack finishes servicing the read request it will pass a new operation structure in the funciton pointed to by cb. Finishe read can be called on the new operation received.

Parameters:

op The operation structure representing this requested io operation.

iovec A pointer to the array of iovecs.

iovec_count The number of iovecs in the array.

wait_for The minimum number of bytes to read before returning... if a driver has no specifc requirement, he should use the user's request... available via GlobusXIOOperationMinimumRead(op)

cb The function to be called when the operation request is completed.

user_arg A user pointer that will be threaded through to the callback.

6.4.3.10 void globus_xio_driver_finished_read (globus_xio_operation_t op, globus_result_t result, globus_size_t nread)

Finished Read.

This function is called to signal globus_xio that the requested read operation has been completed.

Parameters:

op The operation structure representing the requested read operation.

result Return status of the completed operation

nread The number of bytes read

6.4.3.11 void globus_xio_driver_set_eof_received (globus_xio_operation_t op)

EOF state manipulation.

This function is used by drivers that allow multiple outstanding reads at a time. It can only be called on behalf of a read operation (while in the read interface call or the pass_read callback).

Typical use for this would be to hold a driver specific lock and call this when an internal eof has been received. The read operation this is called on behalf of must be finished with an eof error or the results are undefined.

In general, you should not have an eof flag in your driver. Use this call and globus_xio_driver_eof_received() instead. This is necessary to support xio's automatic eof resetting. If your driver absolutely can not be read after an eof has been set, then you will need your own eof flag.

This call will typically only be used just before a finished_read() call.

Parameters:

op The operation structure representing the requested read operation.

6.4.3.12 globus_bool_t globus_xio_driver_eof_received (globus_xio_operation_t op)

EOF state checking.

This function is used by drivers that allow multiple outstanding reads at a time. It can only be called on behalf of a read operation (while in the read interface call or the pass_read callback).

Typical use for this would be to hold a driver specific lock (the same one used when calling globus_xio_driver_set_eof_received()) and call this to see if an eof has been received. If so, the operation should immediately be finished with an eof error (do not _return_ an eof error).

This call will typically only be used in the read interface call.

Parameters:

op The operation structure representing the requested read operation.

Returns:

GLOBUS_TRUE if eof received, GLOBUS_FALSE otherwise.

6.4.3.13 globus_result_t globus_xio_driver_pass_write (globus_xio_operation_t op, globus_xio_iovec_t * iovec, int iovec_count, globus_size_t wait_for, globus_xio_driver_data_callback_t cb, void * user_arg)

Driver write.

This function passes a write operation down the driver stack. After this function is called the op structure is no longer valid. However when the driver stack finishes servicing the write request it will pass a new operation structure in the funciton pointed to by cb. Finished write can be called on the new operation received.

Parameters:

op The operation structure representing this requested io operation.

iovec A pointer to the array of iovecs.

iovec_count The number of iovecs in the array.

wait_for The minimum number of bytes to write before returning... if a driver has no specifc requirement, he should use the user's request... available via GlobusXIOOperationMinimumWrite(op)

cb The function to be called when the operation request is completed.

user_arg A user pointer that will be threaded through to the callback.

6.4.3.14 void globus_xio_driver_finished_write (globus_xio_operation_t op, globus_result_t result, globus_size_t nwritten)

Finished Write.

This function is called to signal globus_xio that the requested write operation has been completed.

Parameters:

op The operation structure representing the requested write operation.

result Return status of the completed operation

nwritten The number of bytes written

6.4.3.15 globus_result_t globus_xio_driver_merge_operation (globus_xio_operation_t top_op, globus_xio_operation_t bottom_op)

Finishes an operation and merge two op structures.

(XXX not implemented yet)

This function will join to operations together and signal globus_xio that it has completed. This is an advanced function. Most drivers will not require its use. This function takes an operation that was created by this driver and passed on to drivers lower on the stack and an operation that came in on the interface function (that has seen the top half of the stack) and joins them together. The purpose of this function is to join data descriptors that were prestaged and cached with those that have later come in at the users request. See the read ahead doc for more information.

Parameters:

- top_op The operation that has seen the top part of the driver stack.
- bottom_op The operation that has seen the bottom part of the driver stack.

(result is always success in this case.. if there is an error, use the other nish() call)

6.5 Driver Programming: String options

A driver can choose to expose parameters as in a string form.

Functions

- globus_result_t globus_xio_string_cntl_bouncer (globus_xio_driver_attr_cntl_t cntl_func, void attr, int cmd,...)
- globus_result_t globus_xio_string_cntl_bool (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)
- globus_result_t globus_xio_string_cntl_ oat (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)
- globus_result_t globus_xio_string_cntl_int (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)
- globus_result_t globus_xio_string_cntl_string (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)
- globus_result_t globus_xio_string_cntl_int_int (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)

6.5.1 Detailed Description

A driver can choose to expose parameters as in a string form.

Providing this feature makes dynamicly setting driver speci c options much easier. a user can then load the driverby name and set speci c options by name all at runtime with no object module references. For example, a TCP driver can be loaded with the string: tcp, and the options can be set with:

port=50668keepalive=yes::nodelay=N

this would set the port to 50668, keepalive to true and nodelay to false. The particular string de nition is de ned by the tcp driver by properly creating a globus_xio_attr_parse_table_t array. Each element of the array is 1 options. There are 3 members of each array entry: key, cmd, and parse function. The key is a string that de nes what option is to be set. In the above example string "port" would be 1 key. cmd tells the driver what cntl is associated with the key. In otherwords, once the string is parsed out what driver speci c control must be called to set the requested option. For more information on controls see Attributes and Cntls. The nal value in the array entry is the parsing function. The pasing function takes the value of the <key>=<value> portion of the string and parses it into data types. once parsed globus_xio_attr_cntl is called and thus the option is set. There are many available parsing functions but the developer is free to right their own if the provided ones are not suf cient. Sample parsing functions follow:

- Driver Programming: String options
- Driver Programming: String options
- Driver Programming: String options
- Driver Programming: String options
- Driver Programming: String options

6.5.2 Function Documentation

6.5.2.1 globus_result_t globus_xio_string_cntl_bouncer (globus_xio_driver_attr_cntl_t cntl_func, void attr, int cmd, ...)

New type functions call this one.

6.5.2.2 globus_result_t globus_xio_string_cntl_bool (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)

String option parsing function.

6.5.2.3 globus_result_t globus_xio_string_cntl_ oat (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)

String option parsing function.

6.5.2.4 globus_result_t globus_xio_string_cntl_int (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)

String option parsing function.

6.5.2.5 globus_result_t globus_xio_string_cntl_string (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)

String option parsing function.

6.5.2.6 globus_result_t globus_xio_string_cntl_int_int (void attr, const char key, const char val, int cmd, globus_xio_driver_attr_cntl_t cntl_func)

String option parsing function.

6.6 Globus XIO File Driver

The File I/O driver.

Modules

- Opening/Closing
- Reading/Writing
- Env Variables

Attributes and Cntls
Types
Error Types

6.6.1 Detailed Description

The File I/O driver.

6.7 Opening/Closing

An XIO handle with the le driver can be created with globus_xio_handle_create()

If there is no handle set on the attr passed to the globus_xio_open() call, it performs the equivalent of an open() call. In this case, the contact string must contain either a pathname or one of stdin://, stdout://, or stderr://. If a pathname is used, that path is opened. If one of the schemes are used the corresponding stdio handle is used (retrieved with leno()).

In either of the above cases, it is most ef cient to call the blocking version of globus_xio_open() It is also safe to call within a locked critical section.

When the XIO handle is closed, the le driver will destroy its internal resources and close the fd (unless this fd was set on an attr or converted from one of the stdio handles).

6.8 Reading/Writing

Both the globus_xio_register_read() and globus_xio_register_write() calls follow similar semantics as described below.

If the waitforbytes parameter is greater than zero, the io will happen asynchronously and be completed when at least waitforbytes has been read/written.

If the waitforbytes parameter is equal to zero, one of the following alternative behaviors occur:

If the length of the buffer is > 0 the read or write happens synchronously. If the user is using one of the blocking xio calls, no internal callback will occur.

If the length of the buffer is also 0, the call behaves like an asynchronous noti cation of data ready to be either read or written. ie, an asynchronous select().

In any case, when an error or EOF occurs before the waitforbytes request has been met, the outgoing nbytes is set to the amount of data actually read/written before the error or EOF occurred.

You may either use GLOBUS_XIO_FILE_SEEK or GLOBUS_XIO_SEEK to position the le pointer before each read or write or you can specify the desired offset on a data descriptor with the xio cmd, GLOBUS_XIO_DD_SET_OFFSET. simultaneous reading and writing is only predictable if the data descriptor method is used.

6.9 Env Variables

The le driver uses the following environment variables

GLOBUS_XIO_FILE_DEBUG Available if using a debug build. See globus_debug.h for format. The File driver de nes the levels TRACE for all function call tracing and INFO for write buffer sizes

GLOBUS_XIO_SYSTEM_DEBUG Available if using a debug build. See globus_debug.h for format. The File driver uses globus_xio_system (along with the TCP and UDP drivers) which de nes the following levels: TRACE for all function call tracing, DATA for data read and written counts, INFO for some special events, and RAW which dumps the raw buffers actually read or written. This can contain binary data, so be careful when you enable it.

6.10 Attributes and Cntls

Enumerations

enum globus_xio_ le _attr_cmd_t f GLOBUS_XIO_FILE_SET_MODE, GLOBUS_XIO_FILE_GET_MODE, GLOBUS_XIO_FILE_SET_FLAGS, GLOBUS_XIO_FILE_GET_FLAGS, GLOBUS_XIO_FILE_SET_TRUNC_OFFSET, GLOBUS_XIO_FILE_GET_TRUNC_OFFSET, GLOBUS_XIO_FILE_SET_HANDLE, GLOBUS_XIO_FILE_GET_HANDLE, GLOBUS_XIO_FILE_SET_BLOCKING_IO, GLOBUS_XIO_FILE_GET_BLOCKING_IO, GLOBUS_XIO_FILE_SEEK g

Functions

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_MODE, int mode)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_MODE, int mode_out)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_TRUNC_OFFSET, globus_off_t offset)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_TRUNC_OFFSET, globus_off_t offset_out)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_HANDLE, globus_xio_system_ le _t handle)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_HANDLE, globus_xio_system_ le _t handle_out)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_GET_HANDLE, globus_xio_system_ le _t handle_out)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globus_bool_t use_blocking_io)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globus_bool_t use_blocking_io)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_BLOCKING_IO, globus_bool_t use_blocking_io_out)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_GET_BLOCKING_IO, globus_bool_t use_blocking_io_out)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_SEEK, globus_off_t in_out_offset, globus_xio_ le _whence_t whence)

6.10.1 Detailed Description

File driver speci c attrs and cntls.

See also:

globus_xio_attr_cntl() , globus_xio_handle_cntl()

6.10.2 Enumeration Type Documentation

6.10.2.1 enum globus_xio_ le _attr _cmd_t

File driver speci c cntls.

Enumeration values:

GLOBUS_XIO_FILE_SET_MODE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_FILE_GET_MODE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_FILE_SET_FLAGS See usage for globus_xio_attr_cntl.

GLOBUS_XIO_FILE_GET_FLAGS See usage for globus_xio_attr_cntl.

GLOBUS_XIO_FILE_SET_TRUNC_OFFSET See usage for globus_xio_attr_cntl.

GLOBUS_XIO_FILE_GET_TRUNC_OFFSET See usage for globus_xio_attr_cntl.

GLOBUS_XIO_FILE_SET_HANDLE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_FILE_GET_HANDLE See usage for globus_xio_attr_cntl, globus_xio_handle_cntl.

GLOBUS_XIO_FILE_SET_BLOCKING_IO See usage for globus_xio_attr_cntl, globus_xio_handle_cntl.

GLOBUS_XIO_FILE_GET_BLOCKING_IO See usage for globus_xio_attr_cntl, globus_xio_handle_cntl.

GLOBUS_XIO_FILE_SEEK See usage for globus_xio_handle_cntl.

6.10.3 Function Documentation

6.10.3.1 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_MODE, int mode)

Set the le create mode.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use this to set the permissions a non-existent le is created with, The default mode is 0644.

Parameters:
mode A bitwise OR of all the modes desired

See also:
globus_xio_ le _mode_t

string opt: mode=<int>

6.10.3.2 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_MODE, int * mode_out)

Get the le create mode.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
mode_out The current mode will be stored here.

6.10.3.3 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_TRUNC_OFFSET, globus_off_t offset)

Set the le truncate offset.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use this in conjunction with the Types ag to truncate a le to a non-zero offset. If the le was larger than offset bytes, the extra data is lost. If the le was shorter or non-existent, it is extended and the extended part reads as zeros. (default is 0)

Parameters:
  offset The desired size of the le.

6.10.3.4 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_TRUNC_OFFSET, globus_off_t * offset_out)

Get the le truncate offset.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
  offset_out The offset will be stored here.

6.10.3.5 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_HANDLE, globus_xio_-system_le_t handle)

Set the le handle to use.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Do not open a new le, use this preopened handle instead.

Parameters:
  handle Use this handle (fd or HANDLE) for the le. Note: close() will not be called on this handle.

6.10.3.6 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_HANDLE, globus_xio_-system_le_t * handle_out)

Get the le handle in use or in attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
  handle_out The le handle (fd or HANDLE) will be stored here. If none is set, GLOBUS_XIO_TCP_INVALID_-HANDLE will be set.

6.10.3.7 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_GET_HANDLE, globus_xio_system_le_t * handle_out)

Get the le handle in use or in attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
  handle_out The le handle (fd or HANDLE) will be stored here. If none is set, GLOBUS_XIO_TCP_INVALID_-HANDLE will be set.

6.10.3.8 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globus_bool_t use_blocking_io)

Enable true blocking io when making globus_xio_read/write() calls.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Note: use with caution. you can deadlock an entire app with this.

Parameters:
    use_blocking_io If GLOBUS_TRUE, true blocking io will be enabled. GLOBUS_FALSE will disable it (default);

6.10.3.9 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globus_bool_t use_blocking_io)

Enable true blocking io when making globus_xio_read/write() calls.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Note: use with caution. you can deadlock an entire app with this.

Parameters:
    use_blocking_io If GLOBUS_TRUE, true blocking io will be enabled. GLOBUS_FALSE will disable it (default);

6.10.3.10 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_BLOCKING_IO, globus_bool_t * use_blocking_io_out)

Get the blocking io status in use or in attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
    use_blocking_io_out The flag will be set here. GLOBUS_TRUE for enabled.

string opt: blocking=<bool>

6.10.3.11 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_GET_BLOCKING_IO, globus_bool_t * use_blocking_io_out)

Get the blocking io status in use or in attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
    use_blocking_io_out The flag will be set here. GLOBUS_TRUE for enabled.

string opt: blocking=<bool>

6.10.3.12 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_SEEK, globus_off_t * in_out_offset, globus_xio_file_whence_t whence)

Reposition read/write file offset.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

in_out_offset Specify the desired offset (according to whence). On success, the actual le offset will be stored here.

whence Specify how offset should be interpreted.

See also:

globus_xio_ le _whence_t , GLOBUS_XIO_SEEK

6.11 Types

De nes

- #de ne GLOBUS_XIO_FILE_INVALID _HANDLE

Enumerations

- enum globus_xio_ le _ ag _t f GLOBUS_XIO_FILE_CREAT = O_CREAT, GLOBUS_XIO_FILE_EXCL = O_EXCL, GLOBUS_XIO_FILE_RDONLY = O_RDONLY, GLOBUS_XIO_FILE_WRONLY = O_WRONLY, GLOBUS_XIO_FILE_RDWR = O_RDWR, GLOBUS_XIO_FILE_TRUNC = O_TRUNC, GLOBUS_XIO_FILE_APPEND= O_APPEND,GLOBUS_XIO_FILE_BINARY = 0, GLOBUS_XIO_FILE_TEXT = 0 g
- enum globus_xio_ le _mode_t f GLOBUS_XIO_FILE_IRWXU = S_IRWXU, GLOBUS_XIO_FILE_IRUSR = S_IRUSR, GLOBUS_XIO_FILE_IWUSR = S_IWUSR, GLOBUS_XIO_FILE_IXUSR = S_IXUSR, GLOBUS_XIO_FILE_IRWXO = S_IRWXO, GLOBUS_XIO_FILE_IROTH = S_IROTH, GLOBUS_XIO_FILE_IWOTH = S_IWOTH, GLOBUS_XIO_FILE_IXOTH = S_IXOTH, GLOBUS_XIO_FILE_IRWXG = S_IRWXG, GLOBUS_XIO_FILE_IRGRP= S_IRGRP,GLOBUS_XIO_FILE_IWGRP = S_IWGRP, GLOBUS_XIO_FILE_IXGRP = S_IXGRP g
- enum globus_xio_ le _whence_t f GLOBUS_XIO_FILE_SEEK_SET = SEEK_SET, GLOBUS_XIO_FILE_SEEK_CUR = SEEK_CUR, GLOBUS_XIO_FILE_SEEK_END = SEEK_END g

6.11.1 De ne Documentation

6.11.1.1 #de ne GLOBUS_XIO _FILE _INVALID _HANDLE

Invalid handle type.

See also:

GLOBUS_XIO_FILE_SET_HANDLE

6.11.2 Enumeration Type Documentation

6.11.2.1 enum globus_xio_ le _ ag _t

File driver open ags.

OR together all the ags you want

See also:

GLOBUS_XIO_FILE_SET_FLAGS

Enumeration values:

GLOBUS_XIO_FILE_CREAT	Create a new le if it doesn't exist (default).
GLOBUS_XIO_FILE_EXCL	Fail if le already exists.
GLOBUS_XIO_FILE_RDONLY	Open for read only.
GLOBUS_XIO_FILE_WRONLY	Open for write only.
GLOBUS_XIO_FILE_RDWR	Open for reading and writing (default).
GLOBUS_XIO_FILE_TRUNC	Truncate le. See also: GLOBUS_XIO_FILE_SET_TRUNC_OFFSET
GLOBUS_XIO_FILE_APPEND	Open le for appending.
GLOBUS_XIO_FILE_BINARY	File is binary (default).
GLOBUS_XIO_FILE_TEXT	File is text.

6.11.2.2 enum globus_xio_ le_mode_t

File driver create mode.

OR these modes together to get the mode you want.

See also:
GLOBUS_XIO_FILE_SET_MODE

NOTE: for Win32, you only have a choice between read-only and read-write. If the chosen mode does not specify writability, the le will be read only

Enumeration values:

GLOBUS_XIO_FILE_IRWXU	User read, write, and execute.
GLOBUS_XIO_FILE_IRUSR	User read.
GLOBUS_XIO_FILE_IWUSR	User write.
GLOBUS_XIO_FILE_IXUSR	User execute.
GLOBUS_XIO_FILE_IRWXO	Others read, write, and execute.
GLOBUS_XIO_FILE_IROTH	Others read.
GLOBUS_XIO_FILE_IWOTH	Others write.
GLOBUS_XIO_FILE_IXOTH	Others execute.
GLOBUS_XIO_FILE_IRWXG	Group read, write, and execute.
GLOBUS_XIO_FILE_IRGRP	Group read.
GLOBUS_XIO_FILE_IWGRP	Group write.
GLOBUS_XIO_FILE_IXGRP	Group execute.

6.11.2.3 enum globus_xio_ le_whence_t

File driver seek options.

See also:
GLOBUS_XIO_FILE_SEEK

Enumeration values:

GLOBUS_XIO_FILE_SEEK_SET	set the le pointer at the speci ed offset
GLOBUS_XIO_FILE_SEEK_CUR	set the le pointer at current position + offset
GLOBUS_XIO_FILE_SEEK_END	set the le pointer at size of le + offest

6.12 Error Types

The File driver is very close to the system code, so most errors reported by it are converted from the system errno. A few of the exceptions are GLOBUS_XIO_ERROR_EOF, GLOBUS_XIO_ERROR_COMMAND, GLOBUS_XIO_-ERROR_CONTACT_STRING, and GLOBUS_XIO_ERROR_CANCELED

See also:
globus_error_errno_match()

6.13 Globus XIO HTTP Driver

This driver implements the HTTP/1.0 and HTTP/1.1 protocols within the Globus XIO framework.

Modules

- Opening/Closing
- Reading/Writing
- Server
- Attributes and Cntls
- Error Types

Data Structures

- struct globus_xio_http_header_t
  HTTP Header.

Enumerations

- enum globus_xio_http_version_t { , GLOBUS_XIO_HTTP_VERSION_1_0, GLOBUS_XIO_HTTP_VERSION_-1_1 }

6.13.1 Detailed Description

This driver implements the HTTP/1.0 and HTTP/1.1 protocols within the Globus XIO framework.

It may be used with the tcp driver for the standard HTTP protcol stack, or may be combined with the gsi driver for a HTTPS implementation.

This implementation supports user-defined HTTP headers, persistent connections, and chunked transfer encoding.

6.13.2 Enumeration Type Documentation

6.13.2.1 enum globus_xio_http_version_t

Valid HTTP versions, used with the GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HTTP_VERSION attribute and the GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION handle control.

Enumeration values:

GLOBUS_XIO_HTTP_VERSION_1_0	HTTP/1.0.
GLOBUS_XIO_HTTP_VERSION_1_1	HTTP/1.1.

6.14 Opening/Closing

An XIO handle with the http driver can be created with either globus_xio_handle_create() or globus_xio_server_register_accept()

If the handle is created with globus_xio_server_register_accept(), then an HTTP service handle will be created when globus_xio_register_open() is called. The XIO application must call one of the functions in the globus_xio_read() family to receive the HTTP request metadata. This metadata will be returned in the data descriptor associated with that first read: the application should use the GLOBUS_XIO_HTTP_GET_REQUEST descriptor cntl to extract this metadata.

If the handle is created with globus_xio_handle_create(), then an HTTP client handle will be created when globus_xio_register_open() is called. HTTP request headers, version and method may be chosen by setting attributes.

6.15 Reading/Writing

The HTTP driver behaves similar to the underlying transport driver with respect to reads and writes with the exception that metadata must be passed to the handle via open attributes on the client side and will be received as data descriptors as part of the first request read or response read.

6.16 Server

The globus_xio_server_create() causes a new transport-specific listener socket to be created to handle new HTTP connections. globus_xio_server_register_accept() will accept a new connection for processing. globus_xio_server_register_close() cleans up the internal resources associated with the http server and calls close on the listener.

Multiple HTTP requests may be read in sequence from an HTTP server. After each request is processed and the response is sent (either by writing the entire entity body as specified by the Content-Length header or by using the GLOBUS_XIO_HTTP_HANDLE_SET_END_OF_ENTITY handle cntl), the next read will contain the metadata related to the next operation. Only one request will be in process at once–the previous request must have sent or received and EOF (whichever is applicable to the request type).

6.17 Attributes and Cntls

Enumerations

- enum globus_xio_http_handle_cmd_t { GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HEADER, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_STATUS_CODE, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_REASON_PHRASE, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION, GLOBUS_XIO_HTTP_HANDLE_SET_END_OF_ENTITY }
- enum globus_xio_http_attr_cmd_t { GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_METHOD, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HTTP_VERSION, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HEADER, GLOBUS_XIO_HTTP_ATTR_DELAY_WRITE_HEADER, GLOBUS_XIO_HTTP_GET_REQUEST, GLOBUS_XIO_HTTP_GET_RESPONSE }

Functions

- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HEADER, const char *header_name, const char *header_value)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_STATUS_CODE, int status)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_REASON_PHRASE, const char *reason)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION, globus_xio_http_version_t version)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_END_OF_ENTITY)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_METHOD, const char method)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HTTP_VERSION, globus_xio_http_version_t version)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HEADER, const char header_name, const char header_value)

6.17.1 Detailed Description

HTTP driver speci c attrs and cntls.

See also:
globus_xio_attr_cntl() , globus_xio_handle_cntl()

6.17.2 Enumeration Type Documentation

6.17.2.1 enum globus_xio_http_handle_cmd_t

HTTP driver speci c cntls.

Enumeration values:
GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HEADER See usage for: globus_xio_handle_cntl.
GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_STATUS_CODE See usage for: globus_xio_handle_cntl.
GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_REASON_PHRASE See usage for: globus_xio_handle_cntl.
GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION See usage for: globus_xio_handle_cntl.
GLOBUS_XIO_HTTP_HANDLE_SET_END_OF_ENTITY See usage for: globus_xio_handle_cntl.

6.17.2.2 enum globus_xio_http_attr_cmd_t

HTTP driver speci c attribute and data descriptor cntls.

Enumeration values:
GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_METHOD See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HTTP_VERSION See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HEADER See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_HTTP_ATTR_DELAY_WRITE_HEADER See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_HTTP_GET_REQUEST See usage for: globus_xio_data_descriptor_cntl.
GLOBUS_XIO_HTTP_GET_RESPONSE See usage for: globus_xio_data_descriptor_cntl.

6.17.3 Function Documentation

6.17.3.1 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HEADER, const char * header_name, const char * header_value)

Set the value of a response HTTP header.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

header_name Name of the HTTP header to set.

header_value Value of the HTTP header

Certain headers will cause changes in how the HTTP protocol will be handled. These include:

Transfer-Encoding: {identity|chunked} Override the default transfer encoding. If a server knows the exact length of the message body, or does not intend to support persistent connections, it may set this header to be "identity".

If this is set to "identity" and any of the following are true, then the connection will be closed after the end of the response is sent:

- A Content-Length header is not present
- The HTTP version is set to "HTTP/1.0"
- The Connection header is set to "close" Attempts to set this to "chunked" with an "HTTP/1.0" client will fail with a GLOBUS_XIO_ERROR_HTTP_INVALID_HEADER error.

Content-Length: 1*Digit

- Provide a content length for the response message. If the "chunked" transfer encoding is being used, then this header will be silently ignored by the HTTP driver.

Connection: close

- The HTTP connection will be closed after the end of the data response is written.

Returns:

This handle control function can fail with

GLOBUS_XIO_ERROR_MEMORY
GLOBUS_XIO_ERROR_PARAMETER
GLOBUS_XIO_ERROR_HTTP_INVALID_HEADER

6.17.3.2 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_STATUS_CODE, int status)

Set the response status code.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

status Value in the range 100-599 which will be used as the HTTP response code, as per RFC 2616.

If this cntl is not called by a server, then the default value of 200 ("Ok") will be used. If this is called on the client-side of an HTTP connection, the handle control will fail with a GLOBUS_XIO_ERROR_PARAMETER error.

Returns:

This handle control function can fail with

GLOBUS_XIO_ERROR_PARAMETER

6.17.3.3 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_REASON_PHRASE, const char * reason)

Set the response reason phrase.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
reason The value of the HTTP response string, as per RFC 2616.

If this cntl is not called by a server, then a default value based on the handle's response status code will be generated. If this is called on the client-side of an HTTP connection, the handle control will fail with a GLOBUS_XIO_ERROR_PARAMETER error.

Returns:
This handle control function can fail with

- GLOBUS_XIO_ERROR_MEMORY
- GLOBUS_XIO_ERROR_PARAMETER

6.17.3.4 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION, globus_xio_http_version_t version)

Set the response HTTP version.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
version The HTTP version to be used in the server response line.

If this cntl is not called by a server, then the default of GLOBUS_XIO_HTTP_VERSION_1_1 will be used, though no HTTP/1.1 features (chunking, persistent connections, etc) will be assumed if the client request was an HTTP/1.0 request. If this is called on the client-side of an HTTP connection, the handle control will fail with GLOBUS_XIO_ERROR_PARAMETER.

Returns:
This handle control function can fail with

- GLOBUS_XIO_ERROR_MEMORY
- GLOBUS_XIO_ERROR_PARAMETER

6.17.3.5 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_END_OF_ENTITY)

Indicate end-of-entity for an HTTP body.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

HTTP clients and servers must call this command to indicate to the driver that the entity-body which is being sent is completed. Subsequent attempts to write data on the handle will fail.

This handle command MUST be called on the client side of an HTTP connection when the HTTP method is OPTIONS, POST, or PUT, or when the open attributes indicate that an entity will be sent. This handle command MUST be called on the server side of an HTTP request connection when the HTTP method was OPTIONS, GET, POST, or TRACE.

6.17.3.6 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_METHOD, const char * method)

Set the HTTP method to use for a client request.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
  method The request method string ("GET", "PUT", "POST", etc) that will be used in the HTTP request.

If this is not set on the target before it is opened, it will default to GET.

This attribute is ignored when opening the server side of an HTTP connection.

Setting this attribute may fail with

- GLOBUS_XIO_ERROR_MEMORY
- GLOBUS_XIO_ERROR_PARAMETER

6.17.3.7 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HTTP_VERSION, globus_xio_http_version_t version)

Set the HTTP version to use for a client request.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
  version The HTTP version to use for the client request.

If the client is using HTTP/1.0 in a request which will send a request message body (such as a POST or PUT), then the client MUST set the "Content-Length" HTTP header to be the length of the message. If this attribute is not present, then the default of GLOBUS_XIO_HTTP_VERSION_1_1 will be used.

This attribute is ignored when opening the server side of an HTTP connection.

6.17.3.8 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HEADER, const char * header_name, const char * header_value)

Set the value of an HTTP request header.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
  header_name Name of the HTTP header to set.
  header_value Value of the HTTP header

Certain headers will cause the HTTP driver to behave differently than normal. This must be called before

  Transfer-Encoding: {identity|chunked} Override the default transfer encoding. If a server knows the exact length of the message body, or does not intend to support persistent connections, it may set this header to be "identity".

  If this is set to "identity" and any of the following are true, then the connection will be closed after the end of the message is sent:

- A Content-Length header is not present
- The HTTP version is set to "HTTP/1.0"
- The Connection header is set to "close" Attempts to set this to "chunked" with an "HTTP/1.0" client will fail with a GLOBUS_XIO_ERROR_HTTP_INVALID_HEADER error.

Content-Length: 1Digit

- Provide a content length for the response message. If the "chunked" transfer encoding is being used, then this header will be silently ignored by the HTTP driver.

Connection: close

- If present in the server response, the connection will be closed after the end of the data response is written. Otherwise, when persistent connections are enabled, the connection maybe left open by the driver. Persistent connections are not yet implemented.

6.18 Error Types

Enumerations

enum globus_xio_http_errors_t { GLOBUS_XIO_HTTP_ERROR_INVALID_HEADER, GLOBUS_XIO_HTTP_ERROR_PARSE, GLOBUS_XIO_HTTP_ERROR_NO_ENTITY, GLOBUS_XIO_HTTP_ERROR_EOF, GLOBUS_XIO_HTTP_ERROR_PERSISTENT_CONNECTION_DROPPED }

6.18.1 Detailed Description

In addition to errors generated by underlying protocol drivers, the XIO HTTP driver defines a few error conditions specific to the HTTP protocol.

See also:
globus_xio_driver_error_match()

6.18.2 Enumeration Type Documentation

6.18.2.1 enum globus_xio_http_errors_t

Error types used to generate errors using the globus_error_generic module.

Enumeration values:

GLOBUS_XIO_HTTP_ERROR_INVALID_HEADER An attempt to set a header which is not compatible with the HTTP version being used.

GLOBUS_XIO_HTTP_ERROR_PARSE Error parsing HTTP protocol.

GLOBUS_XIO_HTTP_ERROR_NO_ENTITY There is no entity body to read or write.

GLOBUS_XIO_HTTP_ERROR_EOF Server side fake EOF.

GLOBUS_XIO_HTTP_ERROR_PERSISTENT_CONNECTION_DROPPED Persistent connection dropped by the server.

6.19 Globus XIO MODE_E Driver

Modules

Opening/Closing

6.20 Opening/Closing

An XIO handle with the mode_e driver can be created with either globus_xio_handle_create() or globus_xio_server_register_accept().

If the handle is created with globus_xio_handle_create(), the contact string passed to ref globus_xio_register_open() call must contain a host name and service/port. The number of streams required can be specified on the attr using Attributes and Cntls (default is one stream). The stack of drivers to be used on the streams can be specified on the attr using Attributes and Cntls (default is a stack containing TCP driver).

When the XIO handle is closed, the mode_e driver will destroy its internal resources and close the stream(s).

6.21 Reading/Writing

Mode E is unidirectional. Clients can only write and the server can only read. The globus_xio_register_read() enforce that the waitforbytes parameter should be one. When multiple transport streams are used between the client and the server, data might not be delivered in order. globus_xio_data_descriptor_cntl() can be used to get the offset of the data.

globus_xio_register_write() does not enforce any restriction on the waitforbytes parameter.

In any case, when an error or EOF occurs before the waitforbytes request has been met, the outgoing nbytes is set to the amount of data actually read/written before the error or EOF occurred.

6.22 Server

globus_xio_server_create() causes a mode_e listener to be created and listened upon. globus_xio_server_register_accept() performs an asynchronous accept(). globus_xio_server_register_close() cleans up the internal resources associated with the mode_e server.

All accepted handles inherit all mode_e specific attributes set in the attr to globus_xio_server_create()

6.23 Env Variables

The mode_e driver uses the following environment variable

GLOBUS_XIO_MODE_E_DEBUG Available if using a debug build. See globus_debug.h for format.

6.24 Attributes and Cntls

Enumerations

enum globus_xio_mode_e_cmd_t { GLOBUS_XIO_MODE_E_SET_STACK, GLOBUS_XIO_MODE_E_GET_STACK, GLOBUS_XIO_MODE_E_SET_NUM_STREAMS, GLOBUS_XIO_MODE_E_GET_NUM_STREAMS, GLOBUS_XIO_MODE_E_SET_OFFSET_READS, GLOBUS_XIO_MODE_E_GET_OFFSET_

READS, GLOBUS_XIO_MODE_E_SET_MANUAL_EODC, GLOBUS_XIO_MODE_E_GET_MANUAL_-EODC, GLOBUS_XIO_MODE_E_SEND_EOD, GLOBUS_XIO_MODE_E_SET_EODC, GLOBUS_XIO_-MODE_E_DD_GET_OFFSET, GLOBUS_XIO_MODE_E_SET_STACK_ATTR, GLOBUS_XIO_MODE_E_-GET_STACK_ATTR }

Functions

- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_STACK, globus_xio_stack-t stack)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_STACK, globus_xio_stack-t ∗stack_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_NUM_STREAMS, int num-streams)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_NUM_STREAMS, int ∗num_-streams_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_OFFSET_READS, globus-bool_t offset_reads)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_OFFSET_READS, globus-bool_t ∗offset_reads_out)
- globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_MODE_E_SEND_EOD, globus-bool_t send_eod)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_MODE_E_SET_EODC, int eod_count)
- globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_MODE_E_DD_GET_OFFSET, globus_off_t ∗offset_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_STACK_ATTR, globus_xio_-attr_t ∗stack_out)

6.24.1 Detailed Description

Mode_e driver specific attrs and cntls.

See also:
globus_xio_attr_cntl() , globus_xio_handle_cntl() , globus_xio_server_cntl() , globus_xio_data_descriptor_cntl()

6.24.2 Enumeration Type Documentation

6.24.2.1 enum globus_xio_mode_e_cmd_t

MODE_E driver specific cntls.

Enumeration values:

GLOBUS_XIO_MODE_E_SET_STACK	See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_MODE_E_GET_STACK	See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_MODE_E_SET_NUM_STREAMS	See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_MODE_E_GET_NUM_STREAMS	See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_MODE_E_SET_OFFSET_READS	See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_MODE_E_GET_OFFSET_READS	See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_MODE_E_SET_MANUAL_EODC	See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_MODE_E_GET_MANUAL_EODC	See usage for: globus_xio_attr_cntl.

GLOBUS_XIO_MODE_E_SEND_EOD See usage for: globus_xio_data_descriptor_cntl.

GLOBUS_XIO_MODE_E_SET_EODC See usage for: globus_xio_handle_cntl.

GLOBUS_XIO_MODE_E_DD_GET_OFFSET See usage for: globus_xio_data_descriptor_cntl.

GLOBUS_XIO_MODE_E_SET_STACK_ATTR See usage for: globus_xio_attr_cntl.

GLOBUS_XIO_MODE_E_GET_STACK_ATTR See usage for: globus_xio_attr_cntl.

6.24.3 Function Documentation

6.24.3.1 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_STACK, globus_xio_stack_t stack)

Set the stack (of xio drivers) to be used for the connection(s).

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Do not create a new ftp client handle, use this handle instead.

Parameters:

stack Specifies the stack to use for the connection(s). Note: this stack will not be destroyed.

6.24.3.2 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_STACK, globus_xio_stack_t * stack_out)

Get the stack on the attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

stack_out The stack will be stored here. If none is set, GLOBUS_NULL will be set.

6.24.3.3 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_NUM_STREAMS, int num_streams)

Set the number of streams to be used between the client and the server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

num_streams Specifies the number of streams to use.

6.24.3.4 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_NUM_STREAMS, int * num_streams_out)

Get the number of streams on the attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

num_streams_out The stream count will be stored here.

6.24.3.5 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_OFFSET_READS, globus_bool_t offset_reads)

Set ag to indicate whether the data read from user would always be preceded by an offset read or not.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

The user can do a read with wait_for_bytes set to zero, to nd the offset of the data that he is going to get in his next read operation

Parameters:
 offset_reads GLOBUS_TRUE to enable offset reads, GLOBUS_FALSE to disable offset reads (default).

6.24.3.6 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_OFFSET_READS, globus_bool_t * offset_reads_out)

Get OFFSET_READS ag on the attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
 offset_reads_out The OFFSET_READS ag will be stored here.

6.24.3.7 globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_MODE_E_SEND_EOD, globus_bool_t send_eod)

Set SEND_EOD ag.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only for data descriptors to write calls.

Parameters:
 send_eod GLOBUS_TRUE to send EOD, GLOBUS_FALSE to not send EOD (default).

6.24.3.8 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_MODE_E_SET_EODC, int eod_count)

Set EOD count.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only if MANUAL_EODC ag is set to GLOBUS_TRUE.

Parameters:
 eod_count speci es the eod count

6.24.3.9 globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_MODE_E_DD_GET_-OFFSET, globus_off_t * offset_out)

Get offset of the next available data.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only if OFFSETREADS is enabled.

Parameters:
  offset_out offset will be stored here

6.24.3.10 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_STACK_ATTR, globus_xio_attr_t * stack_out)

Get the attr that will be used with the stack.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is intended for use with GLOBUS_XIO_MODE_E_SET_STACK.

Parameters:
  stack_out The stack will be stored here. If none is set, GLOBUS_NULL will be set.

6.25 Types

6.26 Error Types

Enumerations

- enum globus_xio_mode_e_error_type_t { GLOBUS_XIO_MODE_E_HEADER_ERROR }

6.26.1 Detailed Description

The errors reported by MODE_E driver include GLOBUS_XIO_ERROR_COMMAND, GLOBUS_XIO_ERROR_MEMORY, GLOBUS_XIO_ERROR_STATE, GLOBUS_XIO_ERROR_PARAMETER, GLOBUS_XIO_ERROR_EOF, GLOBUS_XIO_ERROR_CANCELED, Error Types

See also:
  globus_xio_driver_error_match(), globus_error_errno_match()

6.26.2 Enumeration Type Documentation

6.26.2.1 enum globus_xio_mode_e_error_type_t

MODE_E driver specific error types.

Enumeration values:
  GLOBUS_XIO_MODE_E_HEADER_ERROR Indicates that the mode_e header is erroneous.

6.27 Globus XIO ORDERING Driver

Modules

- Opening/Closing

6.28 Opening/Closing

Ordering driver is a transform driver and thus has to be used on top of a transport driver. An XIO handle with the ordering driver can be created with either globus_xio_handle_create() or globus_xio_server_register_accept().

When the XIO handle is closed, the ordering driver will destroy its internal resources.

6.29 Reading/Writing

Ordering driver does not allow multiple globus_xio_register_read() to be outstanding. This limitation is there to enforce that the users get the read callback in order. There is a known issue in enforcing the order in which read callbacks are delivered with multiple outstanding reads. This limitation does not restrict the use of parallel reads feature provided by the underlying transport driver. Attributes and Cntls on the attr can be used to specify the number of parallel reads. Ordering will have a maximum of this many number of reads outstanding to the driver below it on the stack. It buffers the data read and delivers it to the user in order.

globus_xio_register_write() does not enforce any restriction.

6.30 Env Variables

The ordering driver uses the following environment variable

GLOBUS_XIO_ORDERING_DEBUG Available if using a debug build. See globus_debug.h for format.

6.31 Attributes and Cntls

Enumerations

enum globus_xio_ordering_cmd_t { GLOBUS_XIO_ORDERING_SET_OFFSET, GLOBUS_XIO_ORDERING_SET_MAX_READ_COUNT, GLOBUS_XIO_ORDERING_GET_MAX_READ_COUNT, GLOBUS_XIO_ORDERING_SET_BUFFERING, GLOBUS_XIO_ORDERING_GET_BUFFERING, GLOBUS_XIO_ORDERING_SET_BUF_SIZE, GLOBUS_XIO_ORDERING_GET_BUF_SIZE, GLOBUS_XIO_ORDERING_SET_MAX_BUF_COUNT, GLOBUS_XIO_ORDERING_GET_MAX_BUF_COUNT }

Functions

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_ORDERING_SET_OFFSET, globus_off_t offset)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_SET_MAX_READ_COUNT, int max_read_count)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_GET_MAX_READ_COUNT, int *max_read_count_out)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_SET_BUFFERING, globus_bool_t buffering) globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_GET_BUFFERING, globus_bool_t buffering_out)

6.31.1 Detailed Description

Ordering driver specific attrs and cntls.

See also:
globus_xio_attr_cntl() , globus_xio_handle_cntl()

6.31.2 Enumeration Type Documentation

6.31.2.1 enum globus_xio_ordering_cmd_t

ORDERING driver specific cntls.

Enumeration values:

GLOBUS_XIO_ORDERING_SET_OFFSET	See usage for globus_xio_handle_cntl.
GLOBUS_XIO_ORDERING_SET_MAX_READ_COUNT	See usage for globus_xio_attr_cntl.
GLOBUS_XIO_ORDERING_GET_MAX_READ_COUNT	See usage for globus_xio_attr_cntl.
GLOBUS_XIO_ORDERING_SET_BUFFERING	See usage for globus_xio_attr_cntl.
GLOBUS_XIO_ORDERING_GET_BUFFERING	See usage for globus_xio_attr_cntl.
GLOBUS_XIO_ORDERING_SET_BUF_SIZE	See usage for globus_xio_attr_cntl.
GLOBUS_XIO_ORDERING_GET_BUF_SIZE	See usage for globus_xio_attr_cntl.
GLOBUS_XIO_ORDERING_SET_MAX_BUF_COUNT	See usage for globus_xio_attr_cntl.
GLOBUS_XIO_ORDERING_GET_MAX_BUF_COUNT	See usage for globus_xio_attr_cntl.

6.31.3 Function Documentation

6.31.3.1 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_ORDERING_SET_OFFSET, globus_off_t offset)

Set offset for the next IO operation.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This is not allowed when there is an outstanding IO operation. This operation clears all the buffered data.

Parameters:
offset Specifies the offset to use in the next IO operation.

6.31.3.2 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_SET_MAX_READ_COUNT, int max_read_count)

Set the maximum number of reads that ordering driver can have outstanding on driver(s) below.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
max_read_count Specifies the maximum number of parallel reads (default is 1).

6.31.3.3 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_GET_MAX_READ_COUNT, int * max_read_count_out)

Get the maximum number of parallel reads set on the attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
 max_read_count_out The maximum number of parallel reads allowed will be stored here.

6.31.3.4 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_SET_BUFFERING, globus_bool_t buffering)

This driver can be used in 2 modes; ordering (care about offsets of the data read - underlying transport driver may deliver data out of order - this driver will rearrange data based on the offset and deliver inorder to user) and buffering (do not care about offsets - just buffer the data read abd deliver it when requested).

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This attribute control can be used to enable buffering.

Parameters:
 buffering GLOBUS_TRUE to enable buffering, GLOBUS_FALSE (default) to disable buffering.

6.31.3.5 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_GET_BUFFERING, globus_bool_t * buffering_out)

Get the buffering flag on the attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
 buffering_out Buffering flag will be stored in here.

6.32 Types

6.33 Error Types

Enumerations

- enum globus_xio_ordering_error_type_t { GLOBUS_XIO_ORDERING_ERROR_READ, GLOBUS_XIO_ORDERING_ERROR_CANCEL }

6.33.1 Detailed Description

The errors reported by ORDERING driver include GLOBUS_XIO_ERROR_COMMAND, GLOBUS_XIO_ERROR_MEMORY, GLOBUS_XIO_ERROR_STATE, GLOBUS_XIO_ERROR_CANCELED

See also:
 globus_xio_driver_error_match(), globus_error_errno_match()

6.33.2 Enumeration Type Documentation

6.33.2.1 enum globus_xio_ordering_error_type_t

ORDERING driver specific error types.

Enumeration values:

GLOBUS_XIO_ORDERING_ERROR_READ	Indicates that an error occured in reading data.
GLOBUS_XIO_ORDERING_ERROR_CANCEL	Indicates an error occured in canceling an operation.

6.34 Globus XIO TCP Driver

The IPV4/6 TCP socket driver.

Modules

- Opening/Closing
- Reading/Writing
- Server
- Env Variables
- Attributes and Cntls
- Types
- Error Types

6.34.1 Detailed Description

The IPV4/6 TCP socket driver.

6.35 Opening/Closing

An XIO handle with the tcp driver can be created with either globus_xio_handle_create() or globus_xio_server_register_accept().

If the handle is created with globus_xio_server_register_accept(), the globus_xio_register_open() call does nothing more than initialize the internal handle with the accepted socket.

If the handle is created with globus_xio_handle_create(), and there is no handle set on the attr passed to the globus_xio_register_open() call, it performs the equivalent of an asynchronous connect() call. In this case, the contact string must contain a host name and service/port. Both the hostname and port number can be numeric or symbolic (eg: some.webserver.com:80 or 214.123.12.1:http). If the hostname is symbolic and it resolves to multiple ip addresses, each one will be attempted in succession, until the connect is successful or there are no more addresses.

When the XIO handle is closed, the tcp driver will destroy its internal resources and close the socket (unless this socket was set on an attr). Any write data pending in system buffers will be sent unless the linger option has been set. Any remaining data in recv buffers will be discarded and (on some systems) a connection reset sent to the peer.

6.36 Reading/Writing

Both the globus_xio_register_read() and globus_xio_register_write() calls follow similar semantics as described below.

If the waitforbytes parameter is greater than zero, the io will happen asynchronously and be completed when at least waitforbytes has been read/written.

If the waitforbytes parameter is equal to zero, one of the following alternative behaviors occur:

If the length of the buffer is > 0 the read or write happens synchronously. If the user is using one of the blocking xio calls, no internal callback will occur.

If the length of the buffer is also 0, the call behaves like an asynchronous noti cation of data ready to be either read or written. ie, an asynchronous select().

In any case, when an error or EOF occurs before the waitforbytes request has been met, the outgoing nbytes is set to the amount of data actually read/written before the error or EOF occurred.

6.37 Server

globus_xio_server_create() causes a tcp listener socket to be created and listened upon. globus_xio_server_register_accept() performs an asynchronous accept(). globus_xio_server_register_close() cleans up the internal resources associated with the tcp server and calls close() on the listener socket (unless the socket was set on the server via the attr)

All accepted handles inherit all tcp speci c attributes set in the attr to globus_xio_server_create(), but can be overridden with the attr to globus_xio_register_open().

6.38 Env Variables

The tcp driver uses the following environment variables

GLOBUS_HOSTNAME Used when setting the hostname in the contact string

GLOBUS_TCP_PORT_RANGE Used to restrict anonymous listener ports ex: GLOBUS_TCP_PORT_RANGE=4000,4100

GLOBUS_TCP_PORT_RANGE_STATE_FILE Used in conjunction with GLOBUS_TCP_PORT_RANGE to maintain last used port among many applications making use of the same port range. That last port + 1 will be used as a starting point within the speci ed tcp port range instead of always starting at the beginning. This is really only necessary when a machine is behind a stateful rewall which is holding a port in a different state than the application's machine. See bugzilla.globus.org, bug 1851 for more info. ex: GLOBUS_TCP_PORT_RANGE_STATE_FILE=/tmp/port_state (le will be created if it does not exist)

GLOBUS_TCP_SOURCERANGE Used to restrict local ports used in a connection

GLOBUS_XIO_TCP_DEBUG Available if using a debug build. See globus_debug.h for format. The TCP driver de nes the levels TRACE for all function call tracing and INFO for write buffer sizes

GLOBUS_XIO_SYSTEM_DEBUG Available if using a debug build. See globus_debug.h for format. The TCP driver uses globus_xio_system (along with the File and UDP drivers) which de nes the following levels: TRACE for all function call tracing, DATA for data read and written counts, INFO for some special events, and RAW which dumps the raw buffers actually read or written. This can contain binary data, so be careful when you enable it.

6.39 Attributes and Cntls

Enumerations

enum globus_xio_tcp_cmd_t { GLOBUS_XIO_TCP_SET_SERVICE, GLOBUS_XIO_TCP_GET_SERVICE, GLOBUS_XIO_TCP_SET_PORT, GLOBUS_XIO_TCP_GET_PORT, GLOBUS_XIO_TCP_SET_BACKLOG, GLOBUS_XIO_TCP_GET_BACKLOG, GLOBUS_XIO_TCP_SET_LISTEN_RANGE, GLOBUS_XIO_TCP_GET_LISTEN_RANGE, GLOBUS_XIO_TCP_GET_HANDLE, GLOBUS_XIO_TCP_SET_HANDLE, GLOBUS_XIO_TCP_SET_INTERFACE, GLOBUS_XIO_TCP_GET_INTERFACE, GLOBUS_XIO_TCP_SET_RESTRICT_PORT, GLOBUS_XIO_TCP_GET_RESTRICT_PORT, GLOBUS_XIO_TCP_SET_REUSEADDR, GLOBUS_XIO_TCP_GET_REUSEADDR, GLOBUS_XIO_TCP_SET_NO_IPV6, GLOBUS_XIO_TCP_GET_NO_IPV6, GLOBUS_XIO_TCP_SET_CONNECT_RANGE, GLOBUS_XIO_TCP_GET_CONNECT_RANGE, GLOBUS_XIO_TCP_SET_KEEPALIVE, GLOBUS_XIO_TCP_GET_KEEPALIVE, GLOBUS_XIO_TCP_SET_LINGER, GLOBUS_XIO_TCP_GET_LINGER, GLOBUS_XIO_TCP_SET_OOBINLINE, GLOBUS_XIO_TCP_GET_OOBINLINE, GLOBUS_XIO_TCP_SET_SNDBUF, GLOBUS_XIO_TCP_GET_SNDBUF, GLOBUS_XIO_TCP_SET_RCVBUF, GLOBUS_XIO_TCP_GET_RCVBUF, GLOBUS_XIO_TCP_SET_NODELAY, GLOBUS_XIO_TCP_GET_NODELAY, GLOBUS_XIO_TCP_SET_SEND_FLAGS, GLOBUS_XIO_TCP_GET_SEND_FLAGS, GLOBUS_XIO_TCP_GET_LOCAL_CONTACT, GLOBUS_XIO_TCP_GET_LOCAL_NUMERIC_CONTACT, GLOBUS_XIO_TCP_GET_REMOTE_CONTACT, GLOBUS_XIO_TCP_GET_REMOTE_NUMERIC_CONTACT, GLOBUS_XIO_TCP_AFFECT_ATTR_DEFAULTS, GLOBUS_XIO_TCP_SET_BLOCKING_IO, GLOBUS_XIO_TCP_GET_BLOCKING_IO }

Functions

- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_SERVICE, const char *servicename)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_SERVICE, char **servicename_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_PORT, int listener_port)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_PORT, int *listener_port_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_LISTEN_RANGE, int listener_min_port, int listener_max_port)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_LISTEN_RANGE, int *listener_min_port_out, int *listener_max_port_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_socket_t *handle_out)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_socket_t *handle_out)
- globus_result_t globus_xio_server_cntl (server, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_socket_t *handle_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_HANDLE, globus_xio_system_socket_t handle)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_RESTRICT_PORT, globus_bool_t restrict_port)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_RESTRICT_PORT, globus_bool_t *restrict_port_out)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_KEEPALIVE, globus_bool_t keepalive)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_KEEPALIVE, globus_bool_t *keepalive_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_LINGER, globus_bool_t linger, int linger_time)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_LINGER, globus_bool_t linger, int linger_time)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_LINGER, globus_bool_t *linger_out, int *linger_time_out)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_LINGER, globus_bool_t *linger_out, int *linger_time_out)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_SNDBUF, int sndbuf)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_SNDBUF, int sndbuf_out)
globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_TCP_SET_SEND_FLAGS, int send ags)
globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_TCP_GET_SEND_FLAGS, int send ags_out)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_LOCAL_CONTACT, char contact_string_out)
globus_result_t globus_xio_server_cntl (server, driver, GLOBUS_XIO_TCP_GET_LOCAL_CONTACT, char contact_string_out)

6.39.1 Detailed Description

Tcp driver speci c attrs and cntls.

See also:
globus_xio_attr_cntl() , globus_xio_handle_cntl() , globus_xio_server_cntl() , globus_xio_data_descriptor_cntl()

6.39.2 Enumeration Type Documentation

6.39.2.1 enum globus_xio_tcp_cmd_t

TCP driver speci c cntls.

Enumeration values:

GLOBUS_XIO_TCP_SET_SERVICE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_SERVICE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_SET_PORT See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_PORT See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_SET_BACKLOG See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_BACKLOG See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_SET_LISTEN_RANGE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_LISTEN_RANGE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_HANDLE See usage for globus_xio_attr_cntl, globus_xio_handle_cntl, globus_xio_server_cntl.

GLOBUS_XIO_TCP_SET_HANDLE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_SET_INTERFACE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_INTERFACE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_SET_RESTRICT_PORT See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_RESTRICT_PORT See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_SET_REUSEADDR See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_REUSEADDR See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_SET_NO_IPV6 See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_NO_IPV6 See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_SET_CONNECT_RANGE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_GET_CONNECT_RANGE See usage for globus_xio_attr_cntl.

GLOBUS_XIO_TCP_SET_KEEPALIVE See usage for globus_xio_attr_cntl, globus_xio_handle_cntl.

GLOBUS_XIO_TCP_GET_KEEPALIVE See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_SET_LINGER See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_GET_LINGER See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_SET_OOBINLINE See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_GET_OOBINLINE See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_SET_SNDBUF See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_GET_SNDBUF See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_SET_RCVBUF See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_GET_RCVBUF See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_SET_NODELAY See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_GET_NODELAY See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_SET_SEND_FLAGS See usage for: globus_xio_data_descriptor_cntl.
GLOBUS_XIO_TCP_GET_SEND_FLAGS See usage for: globus_xio_data_descriptor_cntl.
GLOBUS_XIO_TCP_GET_LOCAL_CONTACT See usage for: globus_xio_handle_cntl, globus_xio_server_cntl.
GLOBUS_XIO_TCP_GET_LOCAL_NUMERIC_CONTACT See usage for: globus_xio_handle_cntl, globus_xio_server_cntl.
GLOBUS_XIO_TCP_GET_REMOTE_CONTACT See usage for: globus_xio_handle_cntl.
GLOBUS_XIO_TCP_GET_REMOTE_NUMERIC_CONTACT See usage for: globus_xio_handle_cntl.
GLOBUS_XIO_TCP_AFFECT_ATTR_DEFAULTS See usage for: globus_xio_attr_cntl.
GLOBUS_XIO_TCP_SET_BLOCKING_IO See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
GLOBUS_XIO_TCP_GET_BLOCKING_IO See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.

6.39.3 Function Documentation

6.39.3.1 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_SERVICE, const char * service_name)

Set the tcp service name to bind to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for globus_xio_server_create().

Parameters:
service_name The service name to use when setting up the listener. If the service name cannot be resolved, the port (if one is set) will be used instead.

string opt: port=<string>

6.39.3.2 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_SERVICE, char ** service_name_out)

Get the tcp service name to bind to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
service_name_out A pointer to the service name will be stored here If none is set, NULL will be passed back. Otherwise, the name will be duplicated with strdup() and the user should call free() on it.

6.39.3.3 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_PORT, int listener_port)

Set the tcp port number to bind to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for globus_xio_server_create(). The default port number is 0 (system assigned)

Parameters:

listener_port The port number to use when setting up the listener. If the service name is also set, this will only be used if that can't be resolved.

string opt: port=<int>

6.39.3.4 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_PORT, int * listener_port_out)

Get the tcp port number to bind to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

listener_port_out The port will be stored here.

6.39.3.5 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_LISTEN_RANGE, int listener_min_port, int listener_max_port)

Set the tcp port range to con ne the server to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for globus_xio_server_create() where no speci c service or port has been set. It overrides the range set in the GLOBUS_TCP_PORT_RANGE env variable. If 'restrict port' is true, the server's listening port will be constrained to the range speci ed.

Parameters:

listener_min_port The lower bound on the listener port. (default 0 – no bound)

listener_max_port The upper bound on the listener port. (default 0 – no bound)

See also:

GLOBUS_XIO_TCP_SET_RESTRICT_PORT

string opt: listen_range=<int>,<int>

6.39.3.6 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_LISTEN_RANGE, int * listener_min_port_out, int * listener_max_port_out)

Get the tcp port range on an attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

listener_min_port_out The lower bound will be stored here.

listener_max_port_out The upper bound will be stored here.

6.39.3.7 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_socket_t * handle_out)

Get the tcp socket handle on an attr, handle, or server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

handle_out The tcp socket will be stored here. If none is set, GLOBUS_XIO_TCP_INVALID_HANDLE will be set.

6.39.3.8 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_socket_t * handle_out)

Get the tcp socket handle on an attr, handle, or server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

handle_out The tcp socket will be stored here. If none is set, GLOBUS_XIO_TCP_INVALID_HANDLE will be set.

6.39.3.9 globus_result_t globus_xio_server_cntl (server, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_socket_t * handle_out)

Get the tcp socket handle on an attr, handle, or server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

handle_out The tcp socket will be stored here. If none is set, GLOBUS_XIO_TCP_INVALID_HANDLE will be set.

6.39.3.10 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_HANDLE, globus_xio_system_socket_t handle)

Set the tcp socket to use for a handle or server.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for globus_xio_server_create() or globus_xio_register_open().

Parameters:

handle Use this handle (fd or SOCKET) for the listener or connection. Note: close() will not be called on this handle.

6.39.3.11 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_RESTRICT_PORT, globus_bool_t restrict_port)

Enable or disable the listener or connector range constraints.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only on attrs for globus_xio_server_create() or globus_xio_register_open() This enables or ignores the port range found in the attr or in then env. By default, those ranges are enabled.

Parameters:
restrict_port GLOBUS_TRUE to enable (default), GLOBUS_FALSE to disable.

See also:
GLOBUS_XIO_TCP_SET_LISTEN_RANGE, GLOBUS_XIO_TCP_SET_CONNECT_RANGE

6.39.3.12 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_RESTRICT_PORT, globus_bool_t * restrict_port_out)

Get the restrict port flag.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
restrict_port_out The restrict port flag will be stored here.

6.39.3.13 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_KEEPALIVE, globus_bool_t keepalive)

Enable tcp keepalive.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used on attrs for globus_xio_server_create(), globus_xio_register_open() and with globus_xio_handle_cntl() to determine whether or not to periodically send "keepalive" messages on a connected socket handle. This may enable earlier detection of broken connections.

Parameters:
keepalive GLOBUS_TRUE to enable, GLOBUS_FALSE to disable (default)

string opt: keepalive=<bool>

6.39.3.14 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_KEEPALIVE, globus_bool_t * keepalive_out)

Get the tcp keepalive flag.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
keepalive_out The tcp keepalive flag will be stored here.

6.39.3.15 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_LINGER, globus_bool_t linger, int linger_time)

Set tcp linger.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used on attrs for globus_xio_server_create(), globus_xio_register_open() and with globus_xio_handle_cntl() to determine what to do when data is in the socket's buffer when the socket is closed. If linger is set to true, then the close operation will block until the socket buffers are empty, or the linger_time has expired. If this is enabled, any data remaining after the linger time has expired, will be discarded. If this is disabled, close finishes immediately, but the OS will still attempt to transmit the remaining data.

Parameters:

linger GLOBUS_TRUE to enable, GLOBUS_FALSE to disable (default)

linger_time The time (in seconds) to block at close time if linger is true and data is queued in the socket buffer.

6.39.3.16 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_LINGER, globus_bool_t linger, int linger_time)

Set tcp linger.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used on attrs for globus_xio_server_create(), globus_xio_register_open() and with globus_xio_handle_cntl() to determine what to do when data is in the socket's buffer when the socket is closed. If linger is set to true, then the close operation will block until the socket buffers are empty, or the linger_time has expired. If this is enabled, any data remaining after the linger time has expired, will be discarded. If this is disabled, close finishes immediately, but the OS will still attempt to transmit the remaining data.

Parameters:

linger GLOBUS_TRUE to enable, GLOBUS_FALSE to disable (default)

linger_time The time (in seconds) to block at close time if linger is true and data is queued in the socket buffer.

6.39.3.17 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_LINGER, globus_bool_t * linger_out, int * linger_time_out)

Get the tcp linger flag and time.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

linger_out The linger flag will be stored here.

linger_time_out The linger time will be set here.

6.39.3.18 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_LINGER, globus_bool_t * linger_out, int * linger_time_out)

Get the tcp linger flag and time.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
linger_out The linger ag will be stored here.
linger_time_out The linger time will be set here.

6.39.3.19 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_SNDBUF, int sndbuf)

Set the tcp socket send buffer size.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used on attrs for globus_xio_server_create(), globus_xio_register_open() and with globus_xio_handle_cntl() to set the size of the send buffer used on the socket.

Parameters:
sndbuf The send buffer size in bytes to use. (default is system speci c)

string opt: sndbuf=<formatted int>

6.39.3.20 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_SNDBUF, int sndbuf_out)

Get the tcp send buffer size on the attr or handle.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
sndbuf_out The send buffer size will be stored here.

6.39.3.21 globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_TCP_SET_SEND_FLAGS, int send_ ags)

Set tcp send ags.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only for data descriptors to write calls.

Parameters:
send_ ags The ags to use when sending data.

See also:
globus_xio_tcp_send_ ags_t

6.39.3.22 globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_TCP_GET_SEND_FLAGS, int send_ ags_out)

Get tcp send ags.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
send_ ags_out The ags to use will be stored here.

6.39.3.23 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_LOCAL_-CONTACT, char ** contact_string_out)

Get local socket info.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

contact_string_out A pointer to a contact string for the local end of a connected socket or listener will be stored here. The user should free() it when done with it. It will be in the format: <hostname>:<port>

See also:

globus_xio_server_get_contact_string(), GLOBUS_XIO_GET_LOCAL_CONTACT

6.39.3.24 globus_result_t globus_xio_server_cntl (server, driver, GLOBUS_XIO_TCP_GET_LOCAL_-CONTACT, char ** contact_string_out)

Get local socket info.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

contact_string_out A pointer to a contact string for the local end of a connected socket or listener will be stored here. The user should free() it when done with it. It will be in the format: <hostname>:<port>

See also:

globus_xio_server_get_contact_string(), GLOBUS_XIO_GET_LOCAL_CONTACT

6.40 Types

De nes

- #de ne GLOBUS_XIO_TCP_INVALID_HANDLE

Enumerations

- enum globus_xio_tcp_send ags_t { GLOBUS_XIO_TCP_SEND_OOB = MSG_OOB }

6.40.1 De ne Documentation

6.40.1.1 #de ne GLOBUS_XIO_TCP_INVALID_HANDLE

Invalid handle type.

See also:

GLOBUS_XIO_TCP_SET_HANDLE

6.40.2 Enumeration Type Documentation

6.40.2.1 enum globus_xio_tcp_send_ ags_t

TCP driver speci c types.

Enumeration values:

GLOBUS_XIO_TCP_SEND_OOB Use this with Attributes and Cntls to send a TCP message out of band (Urgent data ag set).

6.41 Error Types

Enumerations

- enum globus_xio_tcp_error_type_t { GLOBUS_XIO_TCP_ERROR_NO_ADDRS }

6.41.1 Detailed Description

The TCP driver is very close to the system code, so most errors reported by it are converted from the system errno. A few of the exceptions are GLOBUS_XIO_ERROR_EOF, GLOBUS_XIO_ERROR_COMMAND, GLOBUS_XIO_-ERROR_CONTACT_STRING, GLOBUS_XIO_ERROR_CANCELED, and Error Types

See also:

globus_xio_driver_error_match(), globus_error_errno_match()

6.41.2 Enumeration Type Documentation

6.41.2.1 enum globus_xio_tcp_error_type_t

TCP driver speci c error types.

Enumeration values:

GLOBUS_XIO_TCP_ERROR_NO_ADDRS Indicates that no IPv4/6 compatible sockets could be resolved for the speci ed hostname.

6.42 Globus XIO UDP Driver

The IPV4/6 UDP socket driver.

Modules

- Opening/Closing
- Reading/Writing
- Env Variables
- Attributes and Cntls
- Types
- Error Types

6.42.1 Detailed Description

The IPV4/6 UDP socket driver.

6.43 Opening/Closing

An XIO handle with the udp driver can be created with globus_xio_handle_create().

The handle can be created in two modes: open server or connected client. If the contact string does not have a host and port, the udp socket will accept messages from any sender. If a host and port is specified, the udp socket will be 'connected' immediately to that host:port. This blocks packets from any sender other than the contact string. A handle that starts out as an open server can later be 'connected' with Attributes and Cntls (presumably after the first message is received from a sender and his contact info is available).

When the XIO handle is closed, the udp driver will destroy its internal resources and close the socket (unless this socket was set on the attr to globus_xio_register_open()).

6.44 Reading/Writing

globus_xio_register_read() semantics:

If the waitforbytes parameter is greater than zero, the read will happen asynchronously and be completed when at least waitforbytes has been read/written.

If the waitforbytes parameter is equal to zero, one of the following alternative behaviors occur:

If the length of the buffer is > 0 the read happens synchronously. If the user is using one of the blocking xio calls, no internal callback will occur.

If the length of the buffer is also 0, the call behaves like an asynchronous notification of data ready to be read. ie, an asynchronous select().

In any case, when an error occurs before the waitforbytes request has been met, the outgoing nbytes is set to the amount of data actually read before the error occurred.

If the handle is not connected, the user should pass in a data descriptor. After the read, this data descriptor will contain the contact string of the sender. The user can either get this contact string with Attributes and Cntls or pass the data descriptor directly to globus_xio_register_write() to send a message back to the sender.

Also, if the handle is not connected, the waitforbytes should probably be 1 to guarantee that only one packet is received and the sender contact isnt overwritten by multiple packets from different senders.

globus_xio_register_write() semantics:

When performing a write, exactly one UDP packet is sent of the entire buffer length. The waitforbytes parameter is ignored. If the entire buffer can not be written, a Error Types error will be returned with nbytes set to the number of bytes actually sent.

If the handle is not 'connected', a contact string must be set in the data descriptor to globus_xio_register_write(). This can either be done explicitly with Attributes and Cntls or implicitly by passing in a data descriptor received from globus_xio_register_read().

The udp write semantics are always synchronous. No blocking or internal callback will occur when using globus_xio_write().

6.45 Env Variables

The udp driver uses the following environment variables

GLOBUS_HOSTNAME Used when setting the hostname in the contact string
GLOBUS_UDP_PORT_RANGE Used to restrict the port the udp socket binds to
GLOBUS_XIO_SYSTEM_DEBUG Available if using a debug build. See globus_debug.h for format. The UDP driver uses globus_xio_system (along with the File and TCP drivers) which defines the following levels: TRACE

for all function call tracing, DATA for data read and written counts, INFO for some special events, and RAW which dumps the raw buffers actually read or written. This can contain binary data, so be careful when you enable it.

6.46 Attributes and Cntls

Enumerations

enum globus_xio_udp_cmd_t { GLOBUS_XIO_UDP_SET_HANDLE, GLOBUS_XIO_UDP_SET_SERVICE, GLOBUS_XIO_UDP_GET_SERVICE, GLOBUS_XIO_UDP_SET_PORT, GLOBUS_XIO_UDP_GET_PORT, GLOBUS_XIO_UDP_SET_LISTEN_RANGE, GLOBUS_XIO_UDP_GET_LISTEN_RANGE, GLOBUS_XIO_UDP_SET_INTERFACE, GLOBUS_XIO_UDP_GET_INTERFACE, GLOBUS_XIO_UDP_SET_RESTRICT_PORT, GLOBUS_XIO_UDP_GET_RESTRICT_PORT, GLOBUS_XIO_UDP_SET_REUSEADDR, GLOBUS_XIO_UDP_GET_REUSEADDR, GLOBUS_XIO_UDP_SET_NO_IPV6, GLOBUS_XIO_UDP_GET_NO_IPV6, GLOBUS_XIO_UDP_GET_HANDLE, GLOBUS_XIO_UDP_SET_SNDBUF, GLOBUS_XIO_UDP_GET_SNDBUF, GLOBUS_XIO_UDP_SET_RCVBUF, GLOBUS_XIO_UDP_GET_RCVBUF, GLOBUS_XIO_UDP_GET_CONTACT, GLOBUS_XIO_UDP_GET_NUMERIC_CONTACT, GLOBUS_XIO_UDP_SET_CONTACT, GLOBUS_XIO_UDP_CONNECT, GLOBUS_XIO_UDP_SET_MULTICAST }

Functions

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_HANDLE, globus_xio_system_socket_t handle)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_SERVICE, const char *service_name)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_SERVICE, char **service_name_out)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_PORT, int listener_port)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_PORT, int *listener_port_out)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_LISTEN_RANGE, int listener_min_port, int listener_max_port)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_LISTEN_RANGE, int *listener_min_port_out, int *listener_max_port_out)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_RESTRICT_PORT, globus_bool_t restrict_port)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_RESTRICT_PORT, globus_bool_t *restrict_port_out)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_HANDLE, globus_xio_system_socket_t *handle_out)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_HANDLE, globus_xio_system_socket_t *handle_out)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_SET_SNDBUF, int sndbuf)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_SNDBUF, int *sndbuf_out)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_CONTACT, char **contact_string_out)
globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_UDP_GET_CONTACT, char **contact_string_out)
globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_UDP_SET_CONTACT, char *contact_string)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_CONNECT, char *contact_string)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_MULTICAST, char *contact_string)

6.46.1 Detailed Description

UDP driver specific attrs and cntls.

See also:
globus_xio_attr_cntl() , globus_xio_handle_cntl() , globus_xio_data_descriptor_cntl()

6.46.2 Enumeration Type Documentation

6.46.2.1 enum globus_xio_udp_cmd_t

UDP driver specific cntls.

Enumeration values:

- GLOBUS_XIO_UDP_SET_HANDLE See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_SET_SERVICE See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_GET_SERVICE See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_SET_PORT See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_GET_PORT See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_SET_LISTEN_RANGE See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_GET_LISTEN_RANGE See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_SET_INTERFACE See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_GET_INTERFACE See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_SET_RESTRICT_PORT See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_GET_RESTRICT_PORT See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_SET_REUSEADDR See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_GET_REUSEADDR See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_SET_NO_IPV6 See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_GET_NO_IPV6 See usage for: globus_xio_attr_cntl.
- GLOBUS_XIO_UDP_GET_HANDLE See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
- GLOBUS_XIO_UDP_SET_SNDBUF See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
- GLOBUS_XIO_UDP_GET_SNDBUF See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
- GLOBUS_XIO_UDP_SET_RCVBUF See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
- GLOBUS_XIO_UDP_GET_RCVBUF See usage for: globus_xio_attr_cntl, globus_xio_handle_cntl.
- GLOBUS_XIO_UDP_GET_CONTACT See usage for: globus_xio_handle_cntl, globus_xio_data_descriptor_cntl.
- GLOBUS_XIO_UDP_GET_NUMERIC_CONTACT See usage for: globus_xio_handle_cntl, globus_xio_data_descriptor_cntl.
- GLOBUS_XIO_UDP_SET_CONTACT See usage for: globus_xio_data_descriptor_cntl.
- GLOBUS_XIO_UDP_CONNECT See usage for: globus_xio_handle_cntl.
- GLOBUS_XIO_UDP_SET_MULTICAST See usage for: globus_xio_attr_cntl.

6.46.3 Function Documentation

6.46.3.1 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_HANDLE, globus_xio_system_socket_t handle)

Set the udp socket to use.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

handle Use this handle (fd or SOCKET). Note: close() will not be called on this handle.

6.46.3.2 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_SERVICE, const char service_name)

Set the udp service name to listen on.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

service_name The service name to use when setting up the listener. If the service name cannot be resolved, the port (if one is set) will be used instead.

6.46.3.3 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_SERVICE, char service_name_out)

Get the service name to listen on.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

service_name_out A pointer to the service name will be stored here If none is set, NULL will be passed back. Otherwise, the name will be duplicated with strdup() and the user should call free() on it.

6.46.3.4 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_PORT, int listener_port)

Set the port number to listen on.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

The default is 0 (system assigned)

Parameters:

listener_port The port number to use when setting up the listener. If the service name is also set, this will only be used if that can't be resolved.

6.46.3.5 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_PORT, int listener_port_out)

the port number to listen on.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
: listener_port_out The port will be stored here.

6.46.3.6 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_LISTEN_RANGE, int listener_min_port, int listener_max_port)

Set the port range to confine the listener to.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used only where no specific service or port has been set. It overrides the range set in the GLOBUS_UDP_PORT_RANGE env variable. If 'restrict port' is true, the listening port will be constrained to the range specified.

Parameters:
: listener_min_port The lower bound on the listener port. (default 0 – no bound)
: listener_max_port The upper bound on the listener port. (default 0 – no bound)

See also:
: GLOBUS_XIO_UDP_SET_RESTRICT_PORT

6.46.3.7 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_LISTEN_RANGE, int listener_min_port_out, int listener_max_port_out)

Get the udp port range on an attr.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:
: listener_min_port_out The lower bound will be stored here.
: listener_max_port_out The upper bound will be stored here.

6.46.3.8 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_RESTRICT_PORT, globus_bool_t restrict_port)

Enable or disable the listener range constraints.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This enables or ignores the port range found in the attr or in then env. By default, those ranges are enabled.

Parameters:
: restrict_port GLOBUS_TRUE to enable (default), GLOBUS_FALSE to disable.

See also:
: GLOBUS_XIO_UDP_SET_LISTEN_RANGE

6.46.3.9 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_RESTRICT_PORT, globus_bool_t restrict_port_out)

Get the restrict port flag.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

restrict_port_out The restrict port flag will be stored here.

6.46.3.10 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_HANDLE, globus_xio_-system_socket_t handle_out)

Get the socket handle on an attr or handle.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

handle_out The udp socket will be stored here. If none is set, GLOBUS_XIO_UDP_INVALID_HANDLE will be set.

6.46.3.11 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_HANDLE, globus_xio_system_socket_t handle_out)

Get the socket handle on an attr or handle.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

handle_out The udp socket will be stored here. If none is set, GLOBUS_XIO_UDP_INVALID_HANDLE will be set.

6.46.3.12 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_SET_SNDBUF, int snd-buf)

Set the socket send buffer size.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Used to set the size of the send buffer used on the socket.

Parameters:

sndbuf The send buffer size in bytes to use. (default is system specific)

6.46.3.13 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_SNDBUF, int sndbuf_out)

Get the send buffer size on the attr or handle.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters:

sndbuf_out The send buffer size will be stored here.

6.46.3.14 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_CONTACT, char contact_string_out)

Get the contact string associated with a handle or data descriptor.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use with globus_xio_handle_cntl() to get a contact string for the udp listener. Use with globus_xio_data_descriptor_cntl() to get the sender's contact string from a data descriptor passed to globus_xio_register_read()

Parameters:

contact_string_out A pointer to a contact string will be stored here. The user should free() it when done with it. It will be in the format: <hostname>:<port>

See also:

GLOBUS_XIO_GET_LOCAL_CONTACT

6.46.3.15 globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_UDP_GET_CONTACT, char contact_string_out)

Get the contact string associated with a handle or data descriptor.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use with globus_xio_handle_cntl() to get a contact string for the udp listener. Use with globus_xio_data_descriptor_cntl() to get the sender's contact string from a data descriptor passed to globus_xio_register_read()

Parameters:

contact_string_out A pointer to a contact string will be stored here. The user should free() it when done with it. It will be in the format: <hostname>:<port>

See also:

GLOBUS_XIO_GET_LOCAL_CONTACT

6.46.3.16 globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_UDP_SET_CONTACT, char contact_string)

Set the destination contact.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Use on a data descriptor passed to globus_xio_register_write() to specify the recipient of the data. This is necessary with unconnected handles or to send to recipients other than the connected one.

Parameters:

contact_string A pointer to a contact string of the format <hostname/ip>:<port/service>

See also:

GLOBUS_XIO_UDP_CONNECT

6.46.3.17 globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_CONNECT, char * contact_string)

Set the default destination contact.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Connecting a handle to a speci c contact blocks packets from any other contact. It also sets the default destination of all outgoing packets so, using Attributes and Cntls is unnecessary.

Parameters:
contact_string A pointer to a contact string of the format <hostname/ip>:<port/service>

6.46.3.18 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_MULTICAST, char * contact_string)

Join a multicast group.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Speci y a multicast group to join. All packets received will be to the speci ed multicast address. Do not use Attributes and Cntls, Attributes and Cntls, or pass a contact string on the open. Consider using Attributes and Cntls to allow other apps to join this group. Use Attributes and Cntls to specify the interface to use. Will not affect handles set with Attributes and Cntls. Attributes and Cntls is ignored.

Parameters:
contact_string A pointer to a contact string of the multicast group to join with the format: <hostname/ip>:<port/service>

6.47 Types

De nes

- #de ne GLOBUS_XIO_UDP_INVALID_HANDLE

6.47.1 De ne Documentation

6.47.1.1 #de ne GLOBUS_XIO_UDP_INVALID_HANDLE

Invalid handle type.

See also:
GLOBUS_XIO_UDP_SET_HANDLE

6.48 Error Types

Enumerations

- enum globus_xio_udp_error_type_t { GLOBUS_XIO_UDP_ERROR_NO_ADDRS, GLOBUS_XIO_UDP_ERROR_SHORT_WRITE }

6.48.1 Detailed Description

The UDP driver is very close to the system code, so most errors reported by it are converted from the system errno. A few of the exceptions are GLOBUS_XIO_ERROR_COMMAND, GLOBUS_XIO_ERROR_CONTACT_STRING, GLOBUS_XIO_ERROR_CANCELED, Error Types, and Error Types

See also:

globus_xio_driver_error_match(), globus_error_errno_match()

6.48.2 Enumeration Type Documentation

6.48.2.1 enum globus_xio_udp_error_type_t

UDP driver specific error types.

Enumeration values:

GLOBUS_XIO_UDP_ERROR_NO_ADDRS Indicates that no IPv4/6 compatible sockets could be resolved for the specified hostname.

GLOBUS_XIO_UDP_ERROR_SHORT_WRITE Indicates that a write of the full buffer failed. Possibly need to increase the send buffer size.

7 globus xio Data Structure Documentation

7.1 globus_xio_http_header_t Struct Reference

HTTP Header.

Data Fields

- char * name
- char * value

7.1.1 Detailed Description

HTTP Header.

7.1.2 Field Documentation

7.1.2.1 char * globus_xio_http_header_t::name

Header Name.

7.1.2.2 char * globus_xio_http_header_t::value

Header Value.

8 globus xio File Documentation

8.1 globus_xio_ le _driver.h File Reference

Header le for XIO File Driver.

De nes

- #de ne GLOBUS_XIO_FILE_INVALID_HANDLE

Enumerations

- enum globus_xio_ le _attr_cmd_t f GLOBUS_XIO_FILE_SET_MODE, GLOBUS_XIO_FILE_GET_MODE, GLOBUS_XIO_FILE_SET_FLAGS, GLOBUS_XIO_FILE_GET_FLAGS, GLOBUS_XIO_FILE_SET_-TRUNC_OFFSET, GLOBUS_XIO_FILE_GET_TRUNC_OFFSET, GLOBUS_XIO_FILE_SET_HANDLE, GLOBUS_XIO_FILE_GET_HANDLE, GLOBUS_XIO_FILE_SET_BLOCKING_IO, GLOBUS_XIO_FILE_-GET_BLOCKING_IO, GLOBUS_XIO_FILE_SEEK g
- enum globus_xio_ le _ ag _t f GLOBUS_XIO_FILE_CREAT = O_CREAT, GLOBUS_XIO_FILE_EXCL = O_EXCL, GLOBUS_XIO_FILE_RDONLY = O_RDONLY, GLOBUS_XIO_FILE_WRONLY = O_WRONLY, GLOBUS_XIO_FILE_RDWR = O_RDWR, GLOBUS_XIO_FILE_TRUNC = O_TRUNC, GLOBUS_XIO_-FILE_APPEND = O_APPEND, GLOBUS_XIO_FILE_BINARY = 0, GLOBUS_XIO_FILE_TEXT = 0 g
- enum globus_xio_ le _mode_t f GLOBUS_XIO_FILE_IRWXU = S_IRWXU, GLOBUS_XIO_FILE_IRUSR = S_IRUSR, GLOBUS_XIO_FILE_IWUSR = S_IWUSR, GLOBUS_XIO_FILE_IXUSR = S_IXUSR, GLOBUS_-XIO_FILE_IRWXO = S_IRWXO, GLOBUS_XIO_FILE_IROTH = S_IROTH, GLOBUS_XIO_FILE_IWOTH = S_IWOTH, GLOBUS_XIO_FILE_IXOTH = S_IXOTH, GLOBUS_XIO_FILE_IRWXG = S_IRWXG, GLOBUS_XIO_FILE_IRGRP = S_IRGRP, GLOBUS_XIO_FILE_IWGRP = S_IWGRP, GLOBUS_XIO_FILE_-IXGRP = S_IXGRP g
- enum globus_xio_ le _whence_t f GLOBUS_XIO_FILE_SEEK_SET = SEEK_SET, GLOBUS_XIO_FILE_-SEEK_CUR = SEEK_CUR, GLOBUS_XIO_FILE_SEEK_END = SEEK_END g

Functions

- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_MODE, int mode)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_MODE, int mode_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_TRUNC_OFFSET, globus_off_t off-set)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_TRUNC_OFFSET, globus_off_-t offset_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_HANDLE, globus_xio_system_ le _t handle)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_HANDLE, globus_xio_system_- le _t handle_out)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_GET_HANDLE, globus_xio_-system_ le _t handle_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globus_bool_-t use_blocking_io)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_SET_BLOCKING_IO, globus_-bool_t use_blocking_io)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_FILE_GET_BLOCKING_IO, globus_bool_-t use_blocking_io_out)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_GET_BLOCKING_IO, globus_bool_t use_blocking_io_out)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_FILE_SEEK, globus_off_t in_out_offset, globus_xio_ le _whence_t whence)

8.1.1 Detailed Description

Header le for XIO File Driver.

8.2 globus_xio_http.h File Reference

Globus XIO HTTP Driver Header.

Data Structures

struct globus_xio_http_header_t
HTTP Header.

Enumerations

enum globus_xio_http_handle_cmd_t f GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HEADER, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_STATUS_CODE, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_REASON_PHRASE, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION, GLOBUS_XIO_HTTP_HANDLE_SET_END_OF_ENTITY g
enum globus_xio_http_attr_cmd_t f GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_METHOD, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HTTP_VERSION, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HEADER, GLOBUS_XIO_HTTP_ATTR_DELAY_WRITE_HEADER, GLOBUS_XIO_HTTP_GET_REQUEST, GLOBUS_XIO_HTTP_GET_RESPONSE g
enum globus_xio_http_errors_t f GLOBUS_XIO_HTTP_ERROR_INVALID_HEADER, GLOBUS_XIO_HTTP_ERROR_PARSE, GLOBUS_XIO_HTTP_ERROR_NO_ENTITY, GLOBUS_XIO_HTTP_ERROR_EOF, GLOBUS_XIO_HTTP_ERROR_PERSISTENT_CONNECTION_DROPPED g
enum globus_xio_http_version_t f , GLOBUS_XIO_HTTP_VERSION_1_0, GLOBUS_XIO_HTTP_VERSION_1_1 g

Functions

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HEADER, const char header_name, const char header_value)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_STATUS_CODE, int status)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_REASON_PHRASE, const char reason)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_RESPONSE_HTTP_VERSION, globus_xio_http_version_t version)
globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_HTTP_HANDLE_SET_END_OF_ENTITY)
globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_METHOD, const char method)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HTTP_-VERSION, globus_xio_http_version_t version)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_SET_REQUEST_HEADER, const char header_name, const char header_value)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_DELAY_WRITE_HEADER)

globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_HTTP_GET_REQUEST, char method, char uri, globus_xio_http_version_t http_version, globus_hashtable_t headers)

globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_HTTP_GET_RESPONSE, int status_code, char reason_phrase, globus_xio_http_version_t http_version, globus_hashtable_t headers)

8.2.1 Detailed Description

Globus XIO HTTP Driver Header.

8.2.2 Function Documentation

8.2.2.1 globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_HTTP_ATTR_DELAY_WRITE_-HEADER)

Delay writing HTTP request until first data write.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

If this attribute is present when opening an HTTP handle, the HTTP request will not be sent immediately upon opening the handle. Instead, it will be delayed until the first data write is done. This allows other HTTP headers to be sent after the handle is opened.

This attribute cntl takes no arguments.

8.2.2.2 globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_HTTP_GET_REQUEST, char method, char uri, globus_xio_http_version_t http_version, globus_hashtable_t headers)

Get HTTP Request Information.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns in the passed parameters values concerning the HTTP request. Any of the parameters may be NULL if the application is not interested in that part of the information.

Parameters:

- method Pointer to be set to the HTTP request method (typically GET, PUT, or POST). The caller must not access this value outside of the lifetime of the data descriptor nor free it.
- uri Pointer to be set to the requested HTTP path. The caller must not access this value outside of the lifetime of the data descriptor nor free it.
- http_version Pointer to be set to the HTTP version used for this request.
- headers Pointer to be set to point to a hashtable of globus_xio_http_header_t values, keyed by the HTTP header names. The caller must not access this value outside of the lifetime of the data descriptor nor free it or any values in it.

8.2.2.3 globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_HTTP_GET_RESPONSE, int * status_code, char ** reason_phrase, globus_xio_http_version_t * http_version, globus_hashtable_t * headers)

Get HTTP Response Information.

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns in the passed parameters values concerning the HTTP response. Any of the parameters may be NULL if the application is not interested in that part of the information.

Parameters:

- status_code Pointer to be set to the HTTP response status code (such as 404), as per RFC 2616. The caller must not access this value outside of the lifetime of the data descriptor nor free it or any values in it.
- reason_phrase Pointer to be set to the HTTP response reason phrase (such as Not Found). The caller must not access this value outside of the lifetime of the data descriptor nor free it or any values in it.
- http_version Pointer to be set to the HTTP version used for this request.
- headers Pointer to be set to point to a hashtable of globus_xio_http_header_t values, keyed by the HTTP header names. The caller must not access this value outside of the lifetime of the data descriptor nor free it or any values in it.

8.3 globus_xio_mode_e_driver.h File Reference

Header file for XIO MODE_E Driver.

Enumerations

- enum globus_xio_mode_e_error_type_t { GLOBUS_XIO_MODE_E_HEADER_ERROR }
- enum globus_xio_mode_e_cmd_t { GLOBUS_XIO_MODE_E_SET_STACK, GLOBUS_XIO_MODE_E_GET_STACK, GLOBUS_XIO_MODE_E_SET_NUM_STREAMS, GLOBUS_XIO_MODE_E_GET_NUM_STREAMS, GLOBUS_XIO_MODE_E_SET_OFFSET_READS, GLOBUS_XIO_MODE_E_GET_OFFSET_READS, GLOBUS_XIO_MODE_E_SET_MANUAL_EODC, GLOBUS_XIO_MODE_E_GET_MANUAL_EODC, GLOBUS_XIO_MODE_E_SEND_EOD, GLOBUS_XIO_MODE_E_SET_EODC, GLOBUS_XIO_MODE_E_DD_GET_OFFSET, GLOBUS_XIO_MODE_E_SET_STACK_ATTR, GLOBUS_XIO_MODE_E_GET_STACK_ATTR }

Functions

- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_STACK, globus_xio_stack_t stack)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_STACK, globus_xio_stack_t * stack_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_NUM_STREAMS, int num_streams)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_NUM_STREAMS, int * num_streams_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_SET_OFFSET_READS, globus_bool_t offset_reads)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_OFFSET_READS, globus_bool_t * offset_reads_out)

- globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_MODE_E_SEND_EOD, globus_bool_t send_eod)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_MODE_E_SET_EODC, int eod_count)
- globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_MODE_E_DD_GET_OFFSET, globus_off_t * offset_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_MODE_E_GET_STACK_ATTR, globus_xio_attr_t * stack_out)

8.3.1 Detailed Description

Header file for XIO MODE_E Driver.

8.4 globus_xio_ordering_driver.h File Reference

Header file for XIO ORDERING Driver.

Enumerations

- enum globus_xio_ordering_error_type_t { GLOBUS_XIO_ORDERING_ERROR_READ, GLOBUS_XIO_ORDERING_ERROR_CANCEL }
- enum globus_xio_ordering_cmd_t { GLOBUS_XIO_ORDERING_SET_OFFSET, GLOBUS_XIO_ORDERING_SET_MAX_READ_COUNT, GLOBUS_XIO_ORDERING_GET_MAX_READ_COUNT, GLOBUS_XIO_ORDERING_SET_BUFFERING, GLOBUS_XIO_ORDERING_GET_BUFFERING, GLOBUS_XIO_ORDERING_SET_BUF_SIZE, GLOBUS_XIO_ORDERING_GET_BUF_SIZE, GLOBUS_XIO_ORDERING_SET_MAX_BUF_COUNT, GLOBUS_XIO_ORDERING_GET_MAX_BUF_COUNT }

Functions

- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_ORDERING_SET_OFFSET, globus_off_t offset)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_SET_MAX_READ_COUNT, int max_read_count)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_GET_MAX_READ_COUNT, int * max_read_count_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_SET_BUFFERING, globus_bool_t buffering)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_ORDERING_GET_BUFFERING, globus_bool_t * buffering_out)

8.4.1 Detailed Description

Header file for XIO ORDERING Driver.

8.5 globus_xio_tcp_driver.h File Reference

Header file for XIO TCP Driver.

De nes

- #de ne GLOBUS_XIO_TCP_INVALID_HANDLE

Enumerations

- enum globus_xio_tcp_error_type_t f GLOBUS_XIO_TCP_ERROR_NO_ADDRS g
- enum globus_xio_tcp_cmd_t f GLOBUS_XIO_TCP_SET_SERVICE, GLOBUS_XIO_TCP_GET_SERVICE, GLOBUS_XIO_TCP_SET_PORT, GLOBUS_XIO_TCP_GET_PORT, GLOBUS_XIO_TCP_SET_BACKLOG, GLOBUS_XIO_TCP_GET_BACKLOG, GLOBUS_XIO_TCP_SET_LISTEN_RANGE, GLOBUS_XIO_-TCP_GET_LISTEN_RANGE, GLOBUS_XIO_TCP_GET_HANDLE, GLOBUS_XIO_TCP_SET_HANDLE, GLOBUS_XIO_TCP_SET_INTERFACE, GLOBUS_XIO_TCP_GET_INTERFACE, GLOBUS_XIO_TCP_-SET_RESTRICT_PORT, GLOBUS_XIO_TCP_GET_RESTRICT_PORT, GLOBUS_XIO_TCP_SET_-REUSEADDR, GLOBUS_XIO_TCP_GET_REUSEADDR, GLOBUS_XIO_TCP_SET_NO_IPV6, GLOBUS_-XIO_TCP_GET_NO_IPV6, GLOBUS_XIO_TCP_SET_CONNECT_RANGE, GLOBUS_XIO_TCP_GET_-CONNECT_RANGE, GLOBUS_XIO_TCP_SET_KEEPALIVE, GLOBUS_XIO_TCP_GET_KEEPALIVE, GLOBUS_XIO_TCP_SET_LINGER, GLOBUS_XIO_TCP_GET_LINGER, GLOBUS_XIO_TCP_SET_-OOBINLINE, GLOBUS_XIO_TCP_GET_OOBINLINE, GLOBUS_XIO_TCP_SET_SNDBUF, GLOBUS_-XIO_TCP_GET_SNDBUF, GLOBUS_XIO_TCP_SET_RCVBUF, GLOBUS_XIO_TCP_GET_RCVBUF, GLOBUS_XIO_TCP_SET_NODELAY, GLOBUS_XIO_TCP_GET_NODELAY, GLOBUS_XIO_TCP_SET_-SEND_FLAGS, GLOBUS_XIO_TCP_GET_SEND_FLAGS, GLOBUS_XIO_TCP_GET_LOCAL_CONTACT, GLOBUS_XIO_TCP_GET_LOCAL_NUMERIC_CONTACT, GLOBUS_XIO_TCP_GET_REMOTE_-CONTACT, GLOBUS_XIO_TCP_GET_REMOTE_NUMERIC_CONTACT, GLOBUS_XIO_TCP_AFFECT_-ATTR_DEFAULTS, GLOBUS_XIO_TCP_SET_BLOCKING_IO, GLOBUS_XIO_TCP_GET_BLOCKING_IO g
- enum globus_xio_tcp_send_ ags_t f GLOBUS_XIO_TCP_SEND_OOB = MSG_OOB g

Functions

- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_SERVICE, const char service_-name)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_SERVICE, char service_name_-out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_PORT, int listener_port)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_PORT, int listener_port_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_LISTEN_RANGE, int listener_min_-port, int listener_max_port)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_LISTEN_RANGE, int listener_-min_port_out, int listener_max_port_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_system_-socket_t handle_out)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_-system_socket_t handle_out)
- globus_result_t globus_xio_server_cntl (server, driver, GLOBUS_XIO_TCP_GET_HANDLE, globus_xio_-system_socket_t handle_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_HANDLE, globus_xio_system_-socket_t handle)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_RESTRICT_PORT, globus_bool_t re-strict_port)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_RESTRICT_PORT, globus_bool_t restrict_port_out)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_KEEPALIVE, globus_bool_t keepalive)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_KEEPALIVE, globus_bool_t *keepalive_out)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_SET_LINGER, globus_bool_t linger, int linger_time)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_LINGER, globus_bool_t linger, int linger_time)

globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_TCP_GET_LINGER, globus_bool_t *linger_out, int *linger_time_out)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_LINGER, globus_bool_t *linger_out, int *linger_time_out)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_SET_SNDBUF, int sndbuf)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_SNDBUF, int *sndbuf_out)

globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_TCP_SET_SEND_FLAGS, int send_ ags)

globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_TCP_GET_SEND_FLAGS, int *send_ ags_out)

globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_TCP_GET_LOCAL_CONTACT, char **contact_string_out)

globus_result_t globus_xio_server_cntl (server, driver, GLOBUS_XIO_TCP_GET_LOCAL_CONTACT, char **contact_string_out)

8.5.1 Detailed Description

Header le for XIO TCP Driver.

8.6 globus_xio_udp_driver.h File Reference

Header le for XIO UDP Driver.

De nes

#de ne GLOBUS_XIO_UDP_INVALID_HANDLE

Enumerations

enum globus_xio_udp_error_type_t f GLOBUS_XIO_UDP_ERROR_NO_ADDRS, GLOBUS_XIO_UDP_ERROR_SHORT_WRITE g

enum globus_xio_udp_cmd_t f GLOBUS_XIO_UDP_SET_HANDLE, GLOBUS_XIO_UDP_SET_SERVICE, GLOBUS_XIO_UDP_GET_SERVICE, GLOBUS_XIO_UDP_SET_PORT, GLOBUS_XIO_UDP_GET_PORT, GLOBUS_XIO_UDP_SET_LISTEN_RANGE, GLOBUS_XIO_UDP_GET_LISTEN_RANGE, GLOBUS_XIO_UDP_SET_INTERFACE, GLOBUS_XIO_UDP_GET_INTERFACE, GLOBUS_XIO_UDP_SET_RESTRICT_PORT, GLOBUS_XIO_UDP_GET_RESTRICT_PORT, GLOBUS_XIO_UDP_SET_REUSEADDR, GLOBUS_XIO_UDP_GET_REUSEADDR, GLOBUS_XIO_UDP_SET_NO_IPV6, GLOBUS_XIO_UDP_GET_NO_IPV6, GLOBUS_XIO_UDP_GET_HANDLE, GLOBUS_XIO_UDP_SET_SNDBUF, GLOBUS_XIO_UDP_GET_SNDBUF, GLOBUS_XIO_UDP_SET_RCVBUF, GLOBUS_XIO_UDP_GET_RCVBUF, GLOBUS_XIO_UDP_GET_CONTACT, GLOBUS_XIO_UDP_GET_NUMERIC_CONTACT, GLOBUS_XIO_UDP_SET_CONTACT, GLOBUS_XIO_UDP_CONNECT, GLOBUS_XIO_UDP_SET_MULTICAST g

Functions

- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_HANDLE, globus_xio_system_socket_t handle)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_SERVICE, const char *service_name)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_SERVICE, char **service_name_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_PORT, int listener_port)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_PORT, int *listener_port_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_LISTEN_RANGE, int listener_min_port, int listener_max_port)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_LISTEN_RANGE, int *listener_min_port_out, int *listener_max_port_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_RESTRICT_PORT, globus_bool_t restrict_port)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_RESTRICT_PORT, globus_bool_t *restrict_port_out)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_GET_HANDLE, globus_xio_system_socket_t *handle_out)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_HANDLE, globus_xio_system_socket_t *handle_out)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_SET_SNDBUF, int sndbuf)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_SNDBUF, int *sndbuf_out)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_GET_CONTACT, char **contact_string_out)
- globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_UDP_GET_CONTACT, char **contact_string_out)
- globus_result_t globus_xio_data_descriptor_cntl (dd, driver, GLOBUS_XIO_UDP_SET_CONTACT, char *contact_string)
- globus_result_t globus_xio_handle_cntl (handle, driver, GLOBUS_XIO_UDP_CONNECT, char *contact_string)
- globus_result_t globus_xio_attr_cntl (attr, driver, GLOBUS_XIO_UDP_SET_MULTICAST, char *contact_string)

8.6.1 Detailed Description

Header file for XIO UDP Driver.

9 globus xio Page Documentation

9.1 Data descriptors

globus_xio uses data descriptors to associate meta data with the data being writen or the data read.

Data descriptors flow into the drivers read and write interface functions by way of the operation structure. If the driver is interested in viewing the data decriptor it can request it from the operation structure via a call to globus_xio_driver_operation_get_data_descriptor() and it can view any driver specific data descriptor via a call to globus_xio_driver_data_descriptor_get_specific(). The driver can modify values in the data descriptor by setting values before passing the request down the stack. Several functions are available to modify the data descriptors. There is no need to "set()" the data descriptors back into the operation. The functions for manipluating the values in a DD affect the values xio has directly.

Data descriptors flow back to the driver in the callbacks for the data operations. When calling finished operation on a data operation the driver must pass in a data descriptor. It should get this data descriptor from the io operation callback.

Life Cycle:

Passing in a data descriptor: A data descriptor is first created by the globus_xio user. The user can add driver specific data descriptors to it. Once the usre has created and set the attributes on its data descriptor to their liking they pass it into a globus_xio data operation (either read or write). When the data descriptor is passed on globus_xio will make an internal copy of it. It does this by first coping the user the level data descriptor and then walkinging through the list of driver specific data descriptor contianed in to and requesting the the driver make a copy of the driver specific data descriptor. If ever a driver specific data descriptor is NULL globus_xio need not call into its drivers dd_copy function. If ever the user level data descriptor is NULL globus_xio need not deal with the data descriptor functionality at all.

A data descriptor coming back up the stack Once an io operation reachs the transport driver (the bottom of the stack) it takes on a slightly different role. On the way in it is describing what is requested to be done with the data, on the way out it is describing what has actually been done. Once the transport driver performs the operation it should adjust the data descriptor to reflect what has actually happened (few drivers will need to worry about this). Each driver on the way up can adjust the data descriptor and its driver specific data decriptor. When xio reachs the the top of the stack it calls a user callback. When that callback returns all memory associated with the data descriptor is cleaned up. The interface function globus_xio_driver_data_descriptor_free() is used for this.

9.2 Todo List

Global globus_l_xio_http_accept_callback(globus_xio_operation_t op, globus_result_t result, void *user_arg)
When implemented in the XIO driver framework, parse the request header before returning from this, so the target is populated with meaningful information for the user. This will help enable persistent connections.

Index